# PYTHON *for* BEGINNERS

A Practical Guide For The People Who Want to Learn Python The Right and Simple Way

JOHN SNOWDEN


A Practical Guide For
to Learn Python The F

# PYTHON FOR BEGINNERS

A Practical Guide For The People Who Want to
Learn Python The Right and Simple Way

.

**JOHN SNOWDEN**

# COPYRIGHT

# CONTENTS


**Welcome To Python**
- Advantages Of Becoming A Programmer
- Why Choose Python?

**Programming Languages**
- What Is A Programming Language?
- Programming Languages Structure
- Low And High-Level Languages
- What Languages Exist For Programming?
- The Python Language
- Project: Hello World

**Basic Types**
- Numeric Types
- Boolean Types
- String Types

**Collections**
- Lists
- Tuple
- Set
- Dictionaries

**Flow Control**
- Conditional Control Structures
- Iterative Control Structures

**Functions**
- How To Define A Function In Python
- Control Flow
- Arguments And Parameters

**Modules And Packages**
- Modules

- Packages

**Object-Oriented Programming**
- Elements And Characteristics Of Oop
- Heritage
- Polymorphism
- Method Overload
- Encapsulation

**Functional Programming**
- Higher-Order Functions
- Lambda Functions

**Text Files**
- About Files
- Read From A Text File
- Append Text To An Existing Text File
- Binary Files

**Regular Expressions**
- Metacharacters
- Sets
- Regex Module

**Databases**
- Database Peculiarities
- Creating A Database Table
- Sql Basics

**Error Handling In Python**
- Syntax Errors
- Exceptions
- Records

**Python Web Development**
- Contributions Of Python To Web Development
- Web Frameworks For Python
- Using Django
- Website Project: Blog

## Final Words

# Welcome to Python

Hi! This book is made for all those people who like to work with computers and who want to enter this world at a deeper level through the programming language, which is the means of communication between us and the technology that we use. allows to eliminate the barriers between the digital world and ours, taking control through orders, consecutive actions, data and algorithms that control the physical and logical behavior of a machine. Programming possibilities are unlimited. We can observe it from something as every day as the apps on our cell phone to incredible creations in robotics and artificial intelligence. If you are reading this, it is because you have discovered the potential that programming has as a professional career in a world that, due to various circumstances.

## Advantages of Becoming a Programmer

If you are here out of curiosity, because you have already found some information about the programmer's world, but have not yet decided if you want to master a programming language, I will tell you that being a programmer brings many attractive benefits. Among the reasons that exist to be a programmer, we can mention these ten:

- We are in the best time to be programmers. The technological world is growing all the time, and everyday technology is becoming more important in all types of organizations. It will not go out of style or go anywhere. It is here to stay, and every day, its use will be as natural as possible. This continuous and accelerated growth is easy to observe and important, which guarantees a promising future.
- Therefore, the labor field is quite extensive since there is a great demand for programmers in companies of all kinds, who have understood that only through the use of available technological tools will they be able to add value and compete in the market satisfactorily. In addition, there are many ramifications in

programming, which means that there will always be a field that is ideal for you. Programming does not necessarily have to be tedious. If you find your ideal specialization, it will be something that you always enjoy, which will be a source of pride and satisfaction.

- In addition to being relatively easy to find work, programming projects are well paid worldwide, as there are not enough professionals in this industry yet. This means that salaries are above average, in some cases by a fairly high margin, and it is very likely that your employer or your clients will offer you attractive benefits to retain your talent if they manage to understand each other well.
- You can work from home. Programming does not necessarily have to be carried out in an office environment, since you will be providing the service of creating intangible products, which can be easily shared between computers, avoiding the need for main meetings. In fact, most companies prefer it this way since most are not specifically dedicated to technology, and it is not feasible for them to have plant personnel who are 100% dedicated to it. A freelancer goes hand in hand with a programmer. Tired of complicated bosses? Can't find a schedule that suits your needs? Do you want to start? Scheduling is for you.
- You will exercise your brain since "computational thinking" is excellent for learning to process any type of information properly. You will also to solve problems more efficiently and effectively. Your mind will be much more structured, and that will provide different perspectives. That is, you will see the world with different eyes.
- Being a programmer will awaken your creative side. With very basic knowledge, you can do the most diverse things. A small idea could easily become a great project by contributing to your portfolio or even selling for a large sum of money. As Mark Zuckerberg put it: "Programming allows you to create something new entirely from scratch."
- Scheduling lets you create and take control in a digitized world. You will no longer have to worry about them threatening your privacy or because they sell your data; if you do not want it, you will have enough information to put security locks where you need it.

need it.

- Python is the language of the future. Although we have been educated to focus on traditional languages, such as English, Spanish, French, Chinese, among others, what really keeps everyone connected is the language used by computers.
- You don't need a degree. Programming is about dedication and effort, and with the application of knowledge that you can acquire in a very short time, you can create projects that show your potential clients what you are capable of.
- And finally, what we will show you in this book: anyone can learn! Just like learning a new language, playing an instrument, or practicing a sport, you don't need any specific prior knowledge. All we will require is your determination to become a programmer and follow the steps that we provide in this book with their respective topics and exercises.

If you are already convinced, I invite you to continue reading this book. I promise you that the more you go into each of the topics presented, you will discover all the potential that programming has in a practical way and that you are capable of doing much more than you imagined. Scheduling is not difficult when you invest the right amount of time, are persistent, and value selflearning. You will find that solving the challenges faced during code development is rewarding, and when you can visualize your creations after a day of study, you will feel motivated to continue and eager to know more.

## Why Choose Python?

Let's start by understanding why we choose the programming language that we will be learning. Python 32 is a high-level, clean, elegant, agile, and simple programming language. Unlike many of the great successes that exist in the market, Python was created by Guido Van Rossum, who developed all the components of this language but does not alone receive the merit of what we have today, as thousands of programmers and other anonymous professionals have contributed to its improvement and expansion. The creation of Python occurred under these circumstances according to the words of the creator himself:

*“In December 1989, I was looking for a 'hobby' programming project that would keep me busy during Christmas week. My office (...) would be closed, but I had a computer at home and not much else in my hands. I decided to write an interpreter for the new scripting language I had been thinking of lately: a descendant of ABC that would appeal to Unix / C hackers. I chose Python as a working title for the project, being in a slightly lighter mood. irreverent (and a huge Monty Python's Flying Circus fan).”*

Guido created Python keeping in mind that he wanted this to be an easy and intuitive language, be open-source so that everyone could contribute to its development, be a code as understandable as English itself, and be allowed to develop in a short time to achieve every day usage.

Python has common expressions that make it require fewer lines of code to perform basic tasks than others that exist on the market. Some even define this programming language as minimalist. Python's syntax is very friendly and is its most prominent element.

At first, it was designed for Unix. Later, it was included in other operating systems, so that today we can use it on Windows and Mac OS as well. There will never be a problem with performance as long as the correct interpreter is used. It is also multiparadigm (a paradigm indicates the way to solve a problem) since it allows you to create programs with more than one programming style. Allowing programmers to choose the best paradigm for each project, since all problems require specific solutions, makes it more effective and efficient. It is also multipurpose as it has a multipurpose nature. For example, R is good for data science and Machine Learning, but not in web development. With Python, you can do everything at once.

As if this were not enough, it has a standard library provided by the official Python website, which provides many free resources. It also has a multitude of unofficial libraries, which allows them to execute complex functions more easily than other languages. All of this thanks to the act that it is an open code, which allows the general public to modify it according to their needs and that. In addition, the code is constantly improving, which means that your learning will be low cost, you will be able to work Without the need for a lot of investment each project and therefore, and you will turn

programming into a very profitable activity. You will never be alone! Python has a community of developers looking to contribute, share, and develop new software,

Python brings together the best of all languages through a simple language that offers speed and great performance. For all this, the learning curve will be very short, and in a short time you will be ready to perform in areas such as games, web development, graphic design, scientific computing, data processing, financial applications, artificial intelligence, software development, among others. With Python, everything is possible because it offers the best versatility.

# Programming Languages

## What is a Programming Language?

Let's start by understanding what a language is. This is a very convenient method to express information and later create sequences of actions necessary to perform a task. In our environment, there are two types of language:

- Natural language: Natural language is what we use in our daily lives. That is, it refers to the language we speak (English, Spanish, French, Chinese ...). It has the advantage that we learn it gradually thanks to exposure, observation, and practice, for which it is given the name of natural. However, it has disadvantages, such as limited comprehension since it is difficult to understand people when they speak a different language. There is sometimes ambiguity or imprecision because, in most languages, there are synonyms or the words change their meaning according to their context.
- Symbolic language: It is a set of artificially created symbols to express specific meanings that can be universally understood and avoid ambiguities. Therefore, this type of language can be understood internationally, and each symbol will always have the same meaning regardless of where it is used.

Therefore, a programming language is a clear example of a language that occupies an intermediate position between natural language, used particularly by humans, and the precise symbolic languages that allow us to interact with a machine. In this regard, we should be particularly grateful to the progressive evolution of translation languages that allow us to convert instructions from a programming language to a machine language, making the programming process that we carry out today look the same every time more to natural languages, making the task easier.

Programming languages use Western alphabet characters and numbers as symbols. This set or sets of characters are programmed

by the user and interpreted by the computer. Nowadays, we can distinguish between codes due to their use and popularity: ASCII (American Standard Code for Information Interchange) and EBCDIC (Extended Binary Coded Decimal Interchange Code). The first uses 7 bits for each character to represent, which translates into a total of 27 different characters to represent. In its extended version, it uses 8 bits, making it consist of 256 characters. This code is the standard on all personal computers. On the other hand, the second code always uses 8 bits per character, making it 256 in total. This type of code is used primarily in mainframe computers (commonly used to process data for corporate and scientific research functions) and mid-range computers (business-oriented).

## Programming Languages Structure

A language is made up of a set of symbols and words (vocabulary and lexicon) and a set of rules (syntax and semantics) that allow symbols to be grouped to form the language's sentences. The programming language has a set of special rules that allow it to build a program. We will understand by a program, a set of commands or instructions based on a programming language that a computer interprets to solve a problem or execute a specific function.

Although the terms "programming language" and "computer language" are often used as if they were synonyms, it does not have to be that way, since computer languages encompass programming languages and others, such as HTML (language for the markup of web pages that is not properly a programming language).

A programming language allows one or more programmers to specify precisely what data a computer should operate on, how this data should be stored or transmitted, and what actions it should take under a wide range of circumstances. All this, through a language that tries to be relatively close to human or natural language. A relevant characteristic of programming languages is that more than one programmer can have a common set of instructions that can be understood among them to carry out the construction of the program in a collaborative way.

The basic elements of language are the lexicon, syntax, and semantics. Within the lexicon, we can distinguish the following

components:

- Identifiers: symbolic names that will be given to certain programming elements (e.g., names of variables, types, modules, etc.).
- Constants: data that will not change its value throughout the program.
- Operators: symbols that will represent operations between variables and constants.
- Instructions: special symbols that will represent processing structures and the definition of programming elements.
- Comments: text that will be used to document the programs.

Tthe rules (syntax) or productions specify the symbol sequence that make up a sentence in the language.

Finally, semantics defines the meaning of the syntactic constructions of the language and of the expressions and data types used.

## Low and High-Level Languages

Programming languages can be classified into low-level and high-level languages depending on how close or far they are from the architecture of the machine on which they will operate.

Low-level languages take their fundamentals from the Von Neumann machine, so they are at a level very close to the machine. The instructions are different in each computer, so they are difficult to program and are costly. They are classified into machine language and assembly language.

In contrast, high-level languages are based on abstract machines, and that makes it easier for people to understand them. Their instructions are more flexible and have greater power than the previous language types, but a translator is necessary to convert the program to machine language. Even so, as it does not depend on the processor, the same program works for different computers.

High-level languages have several philosophies for programming, also called paradigms, of which we can highlight the following:

- Imperatives: they are languages controlled by commands or instructions. It is created through statements that, when executed, make the interpreter change the value of one location or more in its memory; in other words, it causes a change of state. The successive change of states is what causes the achievement towards the goal or the creation of a solution. Examples of languages of this type are C, C ++, FORTRAN, ALGOL, PL / I, Pascal, Ada, Smalltalk, and COBOL.
- Applications: Also known as functional, these look at the desired result instead of the available data. This means that the states through which the machine must go to obtain a response are not examined. Rather, the function that must be applied to the state of the machine is identified through access to the set of variables and the creation of combinations in specific forms to get an answer.
- Rule-based languages: They are executed by verifying the presence of a condition that enables the execution of appropriate action. It is similar to an imperative language, with the difference that the sentences are not sequential. Conditions are made with logical expressions. Therefore, they are also known as logical programming languages. Among these languages, Prolog is the main representative.
- Object-oriented programming: Complex data objects are created, and then a limited set of functions are designated to operate on that data. Complex objects are designated as an extension of other simpler objects, inheriting their properties. The programs created with these languages are as efficient as those created with imperatives and are as flexible and reliable as those created with application languages.

Whenever a high-level language is being used, an interpretation or translation process will be necessary, which helps to convert the programming language to machine language so that it can be executed. The process of translating and converting into a program is different depending on the compiler or interpreter used. A compiler is a program that is only responsible for carrying out a translation, not the program, while an interpreter is designed to both translate and execute.

This gives rise to two types of errors. Compilation errors are produced in the compilation or interpretation phase of a program, when the syntactic or semantic rules are not met. Execution errors are produced during the program's execution. These error messages are not produced by the compiler, but by a piece of code that the compiler adds to the program. The third type of error can occur when no reference is made to either of these two. This problem can be observed when the program does not give any error, but the results are not as expected. It may be because the algorithm has been incorrectly implemented or because the algorithm was poorly made.

In short, when we refer to a programming language, we are talking specifically about a set of commands or commands that describe the desired process. Each language has its own instructions and statements, and the combination of both allows us to build computer programs. It is important to emphasize that a programming language is not an application or the program itself, but rather the tool that allows us to create and modify them.

## What Languages Exist for Programming?

In 1945, the mathematician and chemist Jonh (Janos) Von Neumann presented the general principles that a general-purpose machine should follow. The first language in which computers were programmed was that of the processor, that is, instructions analogous to those present in Von Neumann's machine. However, it was necessary to take into account the machine's details in order to perform any calculation, and it was also very tedious to introduce the program into the computer.

In 1951, just seven years after Von Neumann introduced the concept of the stored-in-memory program, Wilkes, Wheeler, and Gill describe a program loader that converts from decimal to binary values to allow for greater convenience in encoding instructions and addresses. In order to simplify programming, assemblers gradually became richer until they became translators of symbolic representations (mnemonics) from machine language (assembly languages) to machine language itself. Assembly languages are still close to machine languages and, although they considerably simplify

the programming process, they maintain two of their main drawbacks.

That is why an attempt was made to create a new language that was not based directly on the machine's own instructions (that did not depend on the specific machine) but rather on an abstraction of these, and that was more comfortable for the programmer. In the same way, physical devices (registers, memory cells, etc.) would not be used directly, but abstractions of these (variables). In this way, a new concept of programming language arose, where each language has an abstract machine associated with which your code can be run.

If we want to run programs written in a high-level language on a specific computer, we must translate them into other equivalents in a specific machine code (manually or through a process called compilation) or have a tool that reads the program and interprets it step by step the meaning of each sentence in the program (interpretation process). Working in this direction, between 1954 and 1958, John Backus led a working group that aimed to carry out a machine code translator of mathematical formulas that would express calculations. The result was both the specification of a high-level language, Fortran, and the realization of a compiler that translated said language into the machine code of a specific computer (IBM 704).

Around 1960, three decisive languages were created: Algol 60, Cobol, and Lisp. In 1958, Lisp, designed by John McCarthy, was a very innovative language in the sense that it is far removed from the Von Neumann concept of the machine. It is based almost exclusively on the use of functions and lambda calculus, and its main purpose was symbolic calculus. It was the precursor of so-called functional languages. This language has been widely used in the field of artificial intelligence. In 1959, the US Department of Defense commissioned the Cobol. Its objective was clearly practical, and although it was carried out without taking into account some of the advances made at the time in the design of programming languages, it was innovative in the treatment of data. It was also the first standardized language, which favored its later use. In 1960, Algol 60 was primarily an academic language where numerous innovative concepts were introduced and adopted by many later languages. The

language was fully specified with the BNF notation, a formalism equivalent to non-contextual grammars. Already in 1964, the Basic was born. This language was designed from the user's point of view. That is, the language was intended to be easy to learn and use. This language was quite successful in teaching and especially in the programming of the first microcomputers, but it was hardly used in the professional environment. Given the above, Simula was developed in 1967, based on the Algol 60.

Pascal was born around 1970 after the software crisis. Niklaus Wirth created a simple and clear language that allowed us to face the increasing complexity of the programs of the time. It is the first language based exclusively on structured programming, and thanks to its simplicity, it has been the ideal language on which to develop the semantics of languages and formal verification, whose beginnings also date back to these years. In addition, Pascal has been a good progenitor of later languages. In this sense, Wirth later designed the Modula-2 language built on many concepts introduced in Pascal, although he emphasizes the construction of the program understood as a set of independent modules. After Pascal, a multitude of languages were developed, including C, which at the cost of a lower level of abstraction, provided great flexibility and control over the machine's resources. The concept of concurrent programming also appeared around this time. The first logic programming language, Prolog, was developed in 1975 by the Kowalski and Colmerauer groups. However, unlike what happened with other languages, its gestation was quite long. It can be considered that the beginnings of Prolog are from 1960. Like Lisp, Prolog is far from the Von Neumann machine's concept, and it is based almost entirely on first-order logic. Prolog has an important advantage over other languages that the verification of programs is almost immediate due to its logical base; however, it has the counterpart that it is usually quite inefficient. it was developed in 1975 by the Kowalski and Colmerauer groups. Prolog has such an important advantage over other languages that the verification of programs is almost immediate due to its logical base; however, its counterpart is usually quite inefficient.

In 1983, Ada was created, a language developed under the US Department of Defense's auspices. It is largely based on Pascal,

although it is more complex. It allows us to adequately address concurrent programming and exception handling and introduces the concept of overload.

Currently, there are a large number of new languages (C ++, Java, Modula-3, Oberon, Delphi, Eiffel ...). Most of the evolutions of those presented here, to which some of the concepts discussed above have been added such as object orientation, exception handling, overhead, modularity, etc.

Python is one of the most widely used programming languages today, and the trend continues to rise. It has it all: it is open-source, simple, and easy to understand syntax, thus saving time and resources. It is one of the best to start with in the world of programming. Python is a versatile language that can have multiple applications such as Artificial Intelligence, thanks to libraries like Keras or TensorFlow. It can also be useful for Big Data applications, due to data processing libraries. This programming language is also used in web development, especially thanks to its Django or Flask frameworks. To give a few examples, the SemRush or Reddit websites are developed with Python.

## The Python Language

It is a high-level programming language (far removed from machine language). It is a platformindependent and object-oriented scripting language prepared to carry out any type of program, from Windows applications to network servers or even web pages. It is an interpreted language, which means that it is not necessary to compile the source code to execute it, which offers advantages such as the speed of development and disadvantages such as lower speed.

The creator of the language is a European named Guido Van Rossum. Van Rossum's goal was to cover the need for a user-friendly object-oriented language that could be used to deal with various tasks within the programming that is usually done in Unix using C. Python’s development lasted several years, during which he worked for various United States companies. By 2000, it already had a fairly complete product and a development team with which it had even partnered on business projects. He currently works at Zope, a

content management platform and application server for the web, of course, completely programmed in Python.

Its general purpose is to create all kinds of programs. It is not a language created specifically for the web, although among its possibilities is the development of pages. Python versions are available on many different computer systems. It was originally developed for Unix, although any system is compatible with the language as long as there is an interpreter programmed for it. It is an interpreted language, which means that the code must not be compiled before its execution. Whe a compilation is done, it is done in a transparent way for the programmer. In certain cases, when code is first executed, some bytecodes are produced that are saved in the system, serving to speed up the implicit compilation that the interpreter performs each time the same code is executed. Python has a command-line interpreter where you can enter statements. Each statement is executed and produces a visible result, which can help us understand the language better and test the results of executing portions of code quickly.

Object-oriented programming is supported in Python; in many cases, it is an easy way to create programs with reusable components. It has many functions incorporated in the language itself, for the treatment of strings, numbers, files, etc. In addition, there are many libraries that we can import into programs to deal with specific topics such as window programming or network systems or things as interesting as creating compressed files in .zip. It is remarkable that Python has a very visual syntax, thanks to an indented notation (with margins) that is mandatory. In many languages, elements such as curly braces or the begin and end keywords are used to separate portions of code. To separate the portions of code in Python, you must tabulate inwards, placing a margin to the code that would go inside a function or a loop. This helps all programmers adopt the same notations and ensures that everyone's programs look very similar.

## Preparing the Work Environment

There are several different implementations of Python: CPython, Jython, IronPython, PyPy, etc. CPython is the most used, the fastest,

and the most mature. When people talk about Python, they usually mean this implementation. In this case, both the interpreter and the modules are written in C. Jython is the Java implementation of Python, while IronPython is its C # (.NET) counterpart. By using these implementations, it is possible to use all the libraries available to Java and .NET programmers. PyPy, lastly, is a Python implementation of Python.

Installing Python is very simple. How can you learn a programming language if you don't have access to it? Python 2 tends to come pre-installed on most Apple computers, but you're better off with Python 3. Python 2 is still used by many companies for one simple reason: they built their websites with Python 2 years ago, and they haven't yet updated to Python 3. Python 3 is a major update to the language, with significant changes that make the transition from 2 to 3 very complicated. That's why many companies that have made their page with Python 2 choose to stay with it. The option is that, or rebuild the entire page. New websites are almost always made with Python 3. In the next few years, companies that have stuck with Python 2 will make the switch to Python 3, since everyone is moving to Python 3,

**Installing Python on Mac OS X**

To start using Python on Mac, we will first resort to the command line. To access this terminal, we can follow two routes:

- Click on Spotlight and type "Terminal."
- Or, open the "Applications" folder, then the "Utilities" folder, and finally "Terminal."

Once the terminal is open, try typing the following command and hitting the Enter key:

```
Jot - 125
```

This should open and print a list of all the numbers from 1 to 25. With this, we have practiced how to use a terminal.

Now we will acquire Homebrew, a manager of packages that will allow us to install and manage software packages written in Python. To obtain it and leave it functional, we will follow the steps below:

1. View the page https://brew.sh
2. Copy the installation command indicated on the page into your terminal. It should look something like this:

/ bin / bash -c "$ (curl -fsSL

https://raw.githubusercontent.com/Homebrew/install/master /install.sh)".

3. A new text will open for you. Once this happens, press the "Enter" key. This will prompt you for a password, type in your computer's password, and press "Enter" again.
4. A new text will open for you. Go to the final lines, and if you read the text “Installation successful,” the process has been a success.

Now we will proceed to install Python; the steps for this will be as follows:

1. Type the following in your terminal:

brew install python3

2. In the new text that will open, the "Summary" line will indicate that you have finished, and it worked.

To check that everything is indeed in order, we will carry out a test as follows:

1. Open your terminal and write the following line to check which version you are working with:

Python3 - version

2. The text you should see after you post the above command is as follows:

Python 3.xx

3. If the above text appears, it means that Python 3 is finally installed and available to be used.

## Installing Python on Windows

To get everything ready, we will carry out three processes. The first corresponds to the Python installation. To achieve this, we will follow the following sequence of steps:

1. Visit the page https://www.python.org/downloads/ to install Python 3.5.2.
2. Click on the installer to start the download.
3. Once the installation starts, be sure to click ADD Python to 3.5 paths.
4. Click install now. If the installation was successful, it will indicate this phrase in the installer.

The following process corresponds to the installation of a more suitable terminal so that we can program. Said terminal would be Git Bash:

1. Download Git & Git Bash through the page https://git-scm.com/download/win
2. Accept the license.
3. Allow it to be installed in the default folder and do not make any changes to the configuration.
4. A screen will appear, indicating that the installation is complete.

To perform a test and verify that everything is in order, you must do the following:

1. Go to the start menu and open Git Bash.
2. Confirm that Python is available by typing: Python - -Version and hitting the Enter key.
3. You should see the text Python 3.xx. If not, it means you have Python 2.7 installed.
4. If you have an older Python installed, uninstall it and restart Git Bash. This way, you should be able to view the previously mentioned text.

Now you must install Pip, a package manager that will allow us to install and manage software packages written in Python:

1. In your terminal, run the following:

```
curl https://bootstrap.pypa.io/get-pip.py> get-pip.py
```

2. Once done, write this other line, which will install Pip:

```
Python get-pip.py
```

Ready! Now we have Python, a terminal, and a manager installed.

## How to Get Python Support

After 20 years of support, Python ended support for Python 2. On January 1, 2020, support for version 2.7 of the Python programming language officially ended. The Python Software Foundation, led by the language's creator, Guido van Rossum, announced that it would no longer receive security updates and bug fixes in the future. This is not something too serious, since the launch of its successor, Python 3, took place no less than 14 years ago, in 2006. In fact, support for Python 2.7 should have ended in 2015. However, the huge popularity of this version (it is still the default version of Python on many Linux distributions) convinced the foundation of the need to support both branches of development and to postpone the "death" date of Python 2.7.

The aforementioned is yet another reason why we suggest you learn Python 3 from scratch. However, you are never alone with either Python 2 or Python 3 since the developer community is large and united. Great software is supported by great people. The user base is enthusiastic, dedicated to encouraging the language's use, and committed to making it diverse and friendly.

The official Python page is https://www.python.org , which provides all the information related to this programming language; among its main contents, you can find:

- Beginners Guide: Which is available both for those who want to start programming from scratch (like you!). to veterans who wish to review or update themselves on this topic.
- Downloads: Even though we have provided you with a very specific guide to get started, you can also explore the Python reads and their corresponding files on the official page.
- Documentation: Here, you will find the standard library, along with additional tutorials and guides. Knowledge is power!
- Community: The phrase on the Python page is, “Python community is vast; diverse & aims to grow; Python is Open." It does justice to what is observed in reality. On the same official page, it is possible to have access to the community, where you can have access to FAQs, conferences, and support groups to solve all the doubts that may arise both in your learning process and in your programming career. Some prominent pages where you can connect with the community are:

    - PYSLACKERS: This is a community open by Python for programming enthusiasts. It contains learning resources, libraries, and resources, rules, and codes shared by the community itself. Meet this community on the page: https://pyslackers.com/web .
    - Python Discord: The place that organizes the community through events and challenges (with prizes!). The ideal place to interact more directly with other developers to learn, obtain resources, collaborate, and solve problems, you will never get stuck. Meet this community on the page: https://pythondiscord.com .
    - Python forum: in this place, topics of all kinds are discussed: new information, tips and advice, projects of all kinds (game development, web development ...), tasks (for those who are studying), and you can even share your own developments to get feedback from other people and improve your project. Meet this community on the page: https://python-forum.io

Your learning will not end with this book; being a programmer is a continuous learning process that each of the projects that you will carry out in the future has its own peculiarities, but you will never be alone or adrift. Python puts at your disposal all the tools you could need along the way. Your community will always offer help to those who have less knowledge or are facing too great a challenge. So if you ever encounter difficulties, don't be discouraged and visit the pages shared here, as they will surely be something of value in your new career as a programmer.

## Project: Hello World

At last, the time you have been waiting for has arrived. The first program that we are going to write in Python is the classic "Hello, world!" And in this language, it is as simple as:

```
print ("Hello world")
```

Run Python and type the above line, and hit Enter. The response you should receive in the console is the text:

```
Hello world
```

What we have done here is use the function built-in print() to print the string Hello, world! On our screen. A string is a sequence of characters. In Python, these are enclosed inside quotes, double quotes, or triple quotes.

Next, we will proceed to create a text file with the previous code so that we can distribute our great little program among our friends. Open your preferred text editor and copy the previous line. Save it as "hello.py," for example.

Running this program is as simple as telling Python the name of the file to run:

```
python hello.py
```

If you use Windows, the .py files will already be associated with the Python interpreter, so doubleclick on the file to run the program. However, as this program does nothing more than print a text on the console, the execution is too fast to be seen. To remedy this, we are going to add a new line that waits for the user to enter data. This

way, a console will display the text "Hello, world!" until we press Enter.

print "Hello world" raw_input ()

We could also run the program from the console as if it were executable in all operating systems: ./hello.py

# Basic Types

Data types are handled in any high-level programming language. Data types define a set of values that have certain characteristics and properties. Let's think for a moment when we were in math class. Surely you had a class in which they taught you the different sets of numbers. The natural ones (1, 2, 3, 4...), the integers (..., -2, -1, 0, 1, 2...), the real ones (... -1.1, -0.3, 2.1...), etc. Well, in programming (and of course in Python), each of those sets would be what we call a data type.

In Python, every value that can be assigned to a variable has a data type associated with it. In Python, everything is an object. So the data types would be the classes (where the properties are defined and what can be done with them), and the variables would be the instances (objects) of the data types. In short, a data type establishes what values a variable can take and what operations can be performed on them.

The basic Python data types are numeric (integer, floating-point, and complex), Booleans, and strings.

### Numeric types

Python defines three basic numeric data types: integers, reals, and complex numbers.

### Integer numbers

Whole numbers are those positive or negative numbers that do not have decimals (other than zero). In Python, they can be represented by the type int or the type long. Python's int type is implemented low-level by a long type of C. And since Python uses underneath C, like C, and unlike Java, the range of values it can represent is platform dependent.

In most machines, the long of C is stored using 32 bits; that is, by using a Python int type variable, we can store numbers from -231 to

231 - 1, or what is the same, from -2,147,483,648 to 2,147,483,647. On 64-bit platforms, the range is -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. Python's long type allows you to store numbers of any precision, being limited only by the memory available on the machine.

When assigning a number to a variable, it will become int unless the number is so large as to require the use of the long type. We can also tell Python that a number is stored using long by adding an L at the end:

# type (whole) return whole long = 23L

We can also represent whole numbers in the binary format *octal or hexadecimal.*

The numbers *octal* are created by prepending “0o” to a sequence of octal digits (from 0 to 7):

ten_octal = 0o12

print (ten_octal)

To create an integer in *hexadecimal* , "0x" must be prepended to a sequence of hexadecimal digits (from 0 to 9 and from A to F).

ten_hex = 0xa

print (ten_hex)

Looking at the numbers in *binary* , "0b" is prepended to a sequence of binary digits (0 and 1).

ten_binary = 0b1010

print (ten_binary)

**Real Numbers**

Real numbers are those with decimals. In Python, they are expressed by the type float. In other programming languages, like C, we also have the type double, similar to float but with higher precision (double = double precision). Python, however, implements its float

type at a low level by means of a double type variable of C, that is, using 64 bits, then in Python double precision is always used, and specifically, the IEEE 754: 1-bit standard is followed for the sign, 11 for the exponent, and 52 for the mantissa. This means that the values that we can represent range from ± 2.2250738585072020 x 10-308 to ± 1.7976931348623157 × 10308. To represent a real number in Python, the integer part is written first, followed by a period and finally the decimal part:

decimal = 0.2703

You can also use scientific notation and add an e (for exponent) to indicate an exponent in base 10. For example: decimal = 0.1e-3

It would be equivalent to 0.1 x 10-3 = 0.1 x 0.001 = 0.0001

## Complex Numbers

Complex numbers are those that have an imaginary part. If you did not know of its existence, it is more than likely that you will never need it, so you can safely skip this section. In fact, most programming languages lack this type, although it is widely used by engineers and scientists in general.

In case you need to use complex numbers, or you are just curious, I will tell you that this type, called complex in Python, is also stored using floating-point because these numbers are an extension of the real numbers. Specifically, it is stored in a C structure, composed of two variables of type double, one of them serving to store the real part and the other for the imaginary part. Complex numbers in Python are represented as follows:

complex = 2.1 + 7.8j

## Boolean Types

In Python, the class that represents boolean values is bool. A Boolean variable can only have two values: True or False. These values are especially important for conditional expressions and loops.

These are the different types of operators with which we can work with Boolean values, the socalled logical or conditional operators:

Boolean values are also the result of expressions that use relational operators (comparisons between values):

| Operator | Description | Example |
|---|---|---|
| and | Is a and b fulfilled? | r = True and False # r is False |
| or | Is a or b? | r = True or False # r is True |
| not | Not | r = not True # r is False |

By default, any object is considered true with two exceptions:

| Operator | Description | Example |
|---|---|---|
| = = | Are a and b equal? | r = 5 = = 3 # r is False |
| ! = | Are they distinctive a and b? | r = 5 =! 3 # r is True |
| < | Is a less than b? | r = 5 = <3 # r is False |
| > | Is a greater than b? | r = 5 => 3 # r is True |

- That implements the method _bool_ (), and this returns False.
- That implements the method _len_ (), and this returns 0.

Furthermore, the following objects/instances are also considered false:

- None.
- False.
- The zero value of any numeric type.
- Sequences and empty collections.

## String Types

Once we have finished with the numbers, it's the letters' turn. Another essential and basic Python building block are the sequences or character strings. This type is known as a string, although its true class is str. Strings are nothing more than text enclosed in single ('string') or double ("string") quotes. Inside the quotation marks, you can add special characters escaping them with \, such as \ n, the new line character, or \ t, the tab character.

A string can be preceded by the character or, or the character r, which indicate, respectively, that it is a string using Unicode encoding and a raw string. Raw strings differ from normal strings in that characters escaped by the backslash (\) are not replaced by their counterparts.

Unicode = u "äóè"

raw = r "\ n"

It is also possible to enclose a string in triple quotes (single or double). This way, we will be able to write the text in several lines, and when printing the string, the line breaks that we entered will be respected without having to resort to the \ n character, as well as the quotation marks without having to escape them.

triple = "" "first line

this will be seen in another line "" "

Strings also support operators such as +, which works by concatenating the strings used as operands, and *, in which the string is repeated as many times as indicated by the number used as the second operand.

a = "on e"

b = "two"

```
c = a + b # c is "onetwo"
c = a * 3 # c is "oneoneone
```

# Collections

A collection of data in programming stores two or more elements in an array with different index numbers, so it helps us to group elements that have something to do with each other. There are four types of data collections in the Python language:

- List: it is an ordered and modifiable collection. Allow duplicate data.
- Tuple: it is an ordered and immutable collection. Allow duplicate data.
- Set: it is a collection that does not have an order or an index. There are no duplicate data.
- Dictionary: it is a collection without order, modifiable, and indexed. It does not allow duplicate data.

When choosing an arrangement type, it is helpful to understand the properties each one possesses. Choosing the right type for a particular data set could mean retention of meaning and increase the efficiency or safety of the program.

**Lists**

The lists can contain any type of data: numbers, strings, Booleans, and also lists. It is an ordered and modifiable collection. In Python, these are declared in brackets:

```
Fruits list = ["Strawberry", "grape", "cherry"]

    Print (fruits_list)
```

If we want to gain access to any element in the list, we will use the following indication, placing the index number (remember that in programming, these start at 0) that we want to print between the brackets.

```
Fruits list = ["Strawberry", "grape", "cherry"]
```

```
    Print (fruits_list)

Fruit list = ["Strawberry", "grape", "cherry"]

    Print (fruits_list [1])
```

The use of square brackets to access and modify the elements of a list is common in many languages, but Python has several very pleasant surprises in store for us. A curios thing about the Python [] operator is that we can also use negative numbers. If a negative number is used as an index, it means that the index starts counting from the end to the left; that is, with [-1] we would access the last element of the list, with [-2] the penultimate, with [-3], the penultimate, and so on.

```
Fruits list = ["Strawberry", "grape", "cherry"]

    Print (fruits_list [-1])
```

Another unusual thing is what in Python is known as slicing or partitioning, and that consists of extending this mechanism to allow selecting portions of the list. If instead of a number, we write two start and end numbers separated by a colon [start: end], Python will interpret that we want a list that goes from the start position to the end position, without including the latter. If we write three numbers (start: end: jump) instead of two, the third is used to determine how many positions to add an element to the list.

```
Fruits list = ["Strawberry", "grape", "cherry", "watermelon",
"cantaloupe", "kiwi", "grapefruit"]
    Print (fruits_list [2: 5])
```

## Tuple

A tuple is a collection of data whose order is unalterable. That is, they are elements ordered in a specific sequence and that have importance. In Python, tuples are enclosed in parentheses. Actually the constructor of the tuple is the comma, not the parentheses, but

the interpreter shows the parentheses, and we should use them for clarity:

t = 1, 2, 3 >>> type (t) type "tuple"

Also, keep in mind that it is necessary to add a comma for tuples of a single element to differentiate it from an element between parentheses.

t = (1)

type (t)

type "int"

t = (1,)

type (t)

type "tuple"

To refer to elements of a tuple, as in a list, use the [] operator:

my_var = t [0] # my_var is 1
my_var = t [0: 2] # my_var is (1, 2)
We can use the [] operator because tuples, like lists, are part of a type of object called sequences. Allow me a small paragraph to indicate that text strings are also sequences, so it will not surprise you that we can do things like these:

c = "Hello world" "

c [0] # hc [5:] # world

c [:: 3] # he llo

Tuples' difference from lists is that tuples do not have these modification mechanisms through the very useful functions that we talked about at the end of the previous section. They are also immutable. That is, their values cannot be modified once created, and they have a fixed size. In exchange for these limitations, tuples are "lighter" than lists, so if the use that we are going to give to a collection is very basic, you can use tuples instead of lists and save memory.

**Set**

The set type in Python is the class used by the language to represent sets. A set is a messy collection of unique elements. That is, they do not repeat themselves. The main characteristic of this data type is that it is a collection whose elements do not keep any order and are also unique. These characteristics mean that the main uses of this class are to know if an element belongs to a collection and to eliminate duplicates of a sequential type (list, tuple, or str).

To create a set, enclose a series of elements in braces {}, or use the class constructor set() and pass it as an argument an iterable object (like a list, a tuple, a string ...).

S = {1,2,3,4}
Python distinguishes this type of operation from creating a dictionary by not including a colon. However, a set cannot include mutable objects such as lists, dictionaries, and even other sets.

## Dictionaries

A dictionary is a collection without order, modifiable, and indexed. In Python, these are enclosed in braces and have keys and values. This means that there will be different types of values within the same category. For example, let's look at a dictionary of movies and directors:

d = {"Love Actually": "Richard Curtis", "Kill Bill": "Tarantino", "Amélie": "Jean-Pierre Jeunet"}

The first value is the key, and the second is the value associated with the key. As a key, we can use any immutable value: we could use numbers, strings, Booleans, tuples ... but not lists or dictionaries since they are mutable. This is because dictionaries are implemented as hash tables, and when entering a new key-value pair in the dictionary, the hash of the key is calculated so that the corresponding entry can be quickly found later. If the key object were modified after it was entered in the dictionary, obviously, its hash would also change and could not be found.

The main difference between dictionaries and lists or tuples is that the values stored in a dictionary are accessed not by their index,

because in fact, they have no order, but by their key, using the [] operator again.

d ["Love Actually"] # return "Richard Curtis"

As in lists and tuples, this operator can also be used to reassign values.

d ["Kill Bill"] = "Quentin Tarantino"

However, in this case, slicing cannot be used, among other things, because dictionaries are not sequences but rather mappings.

# Flow Control

A Python program or script is a set of instructions parsed and executed by the interpreter from top to bottom and from left to right. When all the instructions have been executed, the program ends. However, we have tools to alter the program's natural flow: make a piece of code skip according to this or that condition, repeat a set of instructions, etc.

A control structure is a block of code that allows you to group instructions in a controlled way. To talk about flow control structures in Python, it is essential first to talk about indentation.

In a computer language, indentation is what indents written human language (at the formal level). As for formal language, when one writes a letter, you must respect certain indentations. Computer languages require an indentation. Not all programming languages need an indentation, although it is customary to implement it in order to give the source code greater readability. But in Python's case, the indentation is mandatory since its structure will depend on it.

A control structure, then, is defined as follows:

Control structure opening:

    expressions

Encoding (or coding) is another element of the language that cannot be omitted when talking about control structures. This is nothing more than a directive that is placed at the beginning of a Python file in order to indicate to the system the character encoding used in the file. Utf-8 could be any character encoding. If no character encoding is specified, Python might throw an error if it encounters strange characters:

```
print "En el Ñágara encontré un Ñandú"
```

Instead, indicating the corresponding encoding, the file will be executed successfully:

```
# -*- coding: utf-8 -*-
print "En el Ñágara encontré un Ñandú"
```

In this section, we will talk about two control structures:

- Conditional control structures
- Iterative control structures

## Conditional Control Structures

If a program were nothing more than a list of commands to be executed sequentially, one by one, it would not be very useful. Conditionals allow us to check conditions and make our program behave in one way or another, to execute a piece of code or another, depending on this condition.

Conditional control structures are those that allow us to evaluate if one or more conditions are met, to say what action we are going to execute. The condition evaluation can only return 1 of 2 results: true or false.

In daily life, we act according to the evaluation of conditions, much more frequently than we really think: If the traffic light is green, cross the street. If not, wait for the traffic light to turn green. Sometimes, we also evaluate more than one condition to execute a certain action: If the electricity bill arrives and I have money, pay the bill.

To describe the evaluation to be performed on a condition, relational (or comparison) operators are used:

| Symbol | Meaning | Example | Outcome |
|---|---|---|---|
| = = | Like | 5 = = 7 | False |
| ! = | Other than | Red ! = Green | True |
| <= | Smaller than | 8 <12 | True |
| > = | Greater than | 12> 7 | True |
| <= | Less than or equal to | 12 <= 12 | True |
| > = | Greater than or equal | 4> = 5 | False |

And to evaluate more than one condition simultaneously, logical operators are used:

| Operator | Example | Explanation | Outcome |
|---|---|---|---|
| and | 5 = = 7 and 7 <12 | False and False | False |
| and | 9 <12 and 12> 7 | True and true | True |
| and | 9 <12 and 12> 15 | True and false | False |
| or | 12 = = 12 or 15 <7 | True or false | True |
| or | 7> 5 or 9 <12 | True or true | True |
| xor | 4 = = 4 xor 9> 13 | True or true | False |
| xor | 4 = = xor 9 <3 | True or false | True |

Conditional flow control structures are defined through the use of three reserved keywords from the language: if ,elif Yelse .

**If**

The simplest form of a conditional statement is a if followed by the condition to evaluate, a colon (:), and in the next line and indented, the code to be executed in case this condition is met.

fav = "geekworld.net"

```
# si (if) fav is equal to “geekworld.net” if fav == “geekworld.net”:
print "You have great taste!"
print "Thank you"
```

### If-else

We are now going to see a somewhat more complicated conditional. What would we do if we wanted certain orders to be executed if the condition was not fulfilled? We could certainly add another if that had the negation of the first as a condition:

```
if fav == "geekworld.net":
    print "You have great taste!"
    print "Thank you"
if fav! = "geekworld.net":
    print "Wow, it's a pity"
```

But the conditional has a much more useful second construction. We see in the following example that the second condition can be replaced with an else. If we read the code, we see that it makes a lot of sense: "if fav is equal to mundogeek.net, print this and this; if not, print this other."

```
if fav == "geekworld.net":
    print "You have great taste!"
    print "Thank you"
else:
    print "Wow, it's a pity"
```

### Elif

There is still one more construction to see, which is the one that makes use of the elif. Our last conditional construction to see is the one that makes use of the elif. Unlike the conditional control structures, the iterative ones (also called cyclical or loops) allow us to execute the same code repeatedly, as long as a condition is met.

```
if number <0 :
    print "Negative"
elif number> 0:
    print "Positive"
else:
    print "zero"
```

There is also a construction similar to the operator ?. From other languages, which is nothing more than a compact way of expressing an if-else. In this construction, the predicate C is evaluated, and A is returned if it is true or B if it is not true: A if C else B. Let's see an example:

```
var = "pair" if (num% 2 == 0) else "odd"
```

### Iterative control structures

In Python, there are two cyclic structures:

- Loopwhile
- Loopfor

### While

The while loop executes a code snippet as long as a condition is met. Let's look at the following example:

```
age = 0
while age <18: age =
    age + 1
    print “Congratulations, you have" + str (age)
```

What we observed previously is that the age variable begins with 0. Since the condition that age is less than 18 is true (0 is less than 18), we enter the loop. Age is increased by one, and the message is printed, informing that the user has reached one year.

Now the condition is reevaluated, and one is still less than 18, so the code that increases the age by one year is rerun and prints the age to

the screen. The loop will continue executing until age equals 18, at which point the condition will no longer be met, and the program will continue executing the instructions following the loop.

Now let's imagine that we forgot to write the instruction that increases age. In that case, the condition would never be reached that age was equal to or greater than 18, it would always be 0, and the loop would continue indefinitely writing on the screen. You have reached 0. This is what is known as an infinite loop. However, there are situations where an infinite loop is useful. For example, let's look at a little program that repeats everything the user says until they write goodbye.

```
while True:
    entry = raw_input (">")
    if entry == "bye":
    break
else:
    print entry
```

To obtain what the user writes on the screen, we use the function raw_input. You don't need to know what a function is or how it works exactly. Just accept for now that in each iteration of the loop, the input variable will contain what the user typed until hitting Enter. We then check if what the user wrote was goodbye, in which case the break command is executed or if it was something else, in which case what the user wrote is printed on the screen. The break keyword exits the loop we are in.

Another keyword that we can find inside the loops is continue. As you may have guessed, it does nothing but goes directly to the next iteration of the loop:

```
age = 0
while age <18:
age = age + 1
if age% 2 == 0:
```

```
continue
    print "Congratulations, you have" + str (age)
```

As you can see, this is a small modification of our congratulations program. This time we have added and if that checks if the age is even, we jump to the next iteration instead of printing the message. In other words, with this modification, the program would only print congratulations when the age was odd.

## For

In Python for is used as a generic way to iterate over a sequence. And as such, it tries to facilitate its use for this purpose. This is what a for loop looks like in Python:

```
sequence = ["one", "two", "three"]
for element in sequence:
    print element
```

Let's look at the following example:

```
my_list = ['John', 'Anthony', 'Peter', 'Herbert']
for name in my_list:
    print name
```

Another iteration with the loop for can emulate while . In the following example, for each year in the range 2001 to 2013, print the phrase "Year reports":

```
# - * - coding: utf-8 - * -
for yer in range (2001, 2013):
print "Year reports", str (year)
```

# Functions

Creating functions is inevitable in any type of application. A function is a block of code with an associated name, which receives zero or more arguments as input, follows a sequence of statements which executes the desired operation and returns a value, and/or performs a task. This block can be called when it is needed. Python is a language that gives us a lot of flexibility when creating functions. The use of functions is a very important component of the programming paradigm called structured, and it has several advantages:

- Modularization: allows segmenting a complex program into a series of simpler parts or modules, thus facilitating programming and debugging.
- Reuse: allows the same function to be reused in different programs.

Python already defines by default a set of functions that we can use directly in our applications. You have seen some of them in previous tutorials. For example, the function len (), which gets the number of elements in a container object such as a list, tuple, dictionary, or set. We have also seen the function print (), which displays text on the console.

However, as a programmer, you can define your own functions to structure the code in a way that is more readable and to reuse those parts that are repeated throughout an application. This is a critical task as the number of lines in a program grows.

In principle, a program is an ordered sequence of instructions that are executed one after the other. However, when using functions, you can group parts of those instructions as a smaller unit that executes those instructions and usually returns a result.

**How to define a function in Python**

In Python, functions are declared by typing the keyword def followed by the name of the function and in parentheses, the arguments separated by commas. The def statement is a function definition used to create user-defined function objects. Next, in another line, indented and after the colon, we would have the lines of code that make up the code to be executed by the function:

```
def my_function (param1, param2):
    print param1
    print param2
```

We can also find a text string as the first line of the body of the function. These chains are known by the name of docstring (documentation string) and serve, as the name suggests, as function documentation.

```
def my_function (param1, param2) :
    "" "This function print the two previous values   as parameters ""
    "
    print param1
    print param2
```

It is important to clarify that when declaring the function, all we do is associate a name to the code fragment that makes up the function so that we can execute said code later by referencing it by name. That is, at the time of writing these lines, the function is not executed. To call the function (execute your code), you would write:

```
my_function ("hello", 2)
```

The association of the parameters and the values passed to the function is normally done from left to right: as we have given param1 a "hello" value and param2 is 2, my_function would print hello on one line, and then 2. However, it is possible to modify the order of the parameters if we indicate the name of the parameter to associate the value to when calling the function:

```
my_function (param2 = 2, param1 = "hello")
```

The number of values that are passed as a parameter when calling the function has to match the number of parameters that the function accepts according to the declaration of the function. Otherwise, Python will complain:

```
>>> my_function ("hello")

Traceback (most recent call last):

File "", line 1, in

TypeError: my_function () takes exactly 2 arguments (1 given)
```

To define functions with a variable number of arguments, we put the last parameter for the function whose name must be preceded by a * sign:

```
def various (param1, param2, * others):

    for val in others:

        print val

various (1, 2)

various (1, 2, 3)

various (1, 2, 3, 4)
```

This syntax works by creating a tuple (named others in the example) in which the values of all the extra parameters passed as arguments are stored. For the first call, various (1, 2), the other tuple would be empty since no more parameters than the two defined by default have been passed; therefore, nothing would be printed. In the second call, others would be worth (3), and in the third (3, 4).

You can also precede the name of the last parameter with **, in which case a dictionary would be used instead of a tuple. The keys of this dictionary would be the names of the parameters indicated when calling the function and the values of the dictionary, the values associated with these parameters. The following example uses the dictionary items function, which returns a list of its elements, to print the parameters that the dictionary contains.

```
def various (param1, param2, ** others):
```

```
    for i in others.items ():
        print i
various (1, 2, third = 3)
```

It is important to emphasize that defining is not invoking. Let's look at this program:

```
Def (square (x)
    Return x ** 2
```

If we try to run it, nothing will happen at all; well, at least nothing that appears on the screen. Defining a function just makes Python silently "learn," a calculation method associated with the identifier square. Nothing else. Let's do the test by running the program:

```
$ Python square.py
```

Nothing has been printed on the screen. It is not that there is no print, but that defining a function is a process that does not echo on the screen. We repeat: defining a function only associates a calculation method with an identifier and does not imply executing said calculation method.

## Control Flow

To ensure that a function is defined before its first use, you need to know the order in which the statements are executed; this is called the control flow or flow of execution. Execution always begins with the first statement in the program. Statements are executed one at a time, in order, until a function call is reached. Function definitions do not alter the flow of program execution but remember that the statements within the function are not executed until the function call is made. Although not common, you can define one function within another. In this case, the inner function definition is not executed until the outer function is called.

Function calls are like a detour in the flow of execution. Instead of going to the next statement, the flow jumps to the first line of the called function, executes all the statements it finds there and resumes execution from where it left off. This sounds pretty simple ... until you remember that one function can call another. While we

are in the middle of a function, we might be forced to abandon it and go to execute statements in yet another function. But while we're in this new role, we might as well go out and run yet another role!

Fortunately, Python is good at taking note of where it is so that every time a function completes, the program picks up where it left off in the calling function. When it reaches the end of the program, it ends. What is the moral of this whole story? When you're reading a program, don't read it from top to bottom. Instead, follow the flow of execution.

## Arguments and Parameters

When defining a function, the values which are received are called parameters, but during the call, the values that are sent are called arguments. Some of the internal functions require arguments, the values that control how the function performs its task. For example, if you want to find the sine of a number, you have to indicate what number it is. Thus, sine takes a numeric value as an argument. Some functions take more than one argument, such as pow, which takes two arguments: the base and the exponent. Inside the function, the values passed to it are assigned to variables called parameters. Here's an example of a user-defined function, which takes a parameter:

```
def printDouble (step):
    print step, step
```

This function takes a single argument and assigns it to a parameter called step. The parameter value (at this point, we still have no idea what it will be) is printed twice, followed by a newline character. The name step was chosen to suggest that the name you give to a parameter is up to you, but in general, it is better to choose a more illustrative name than step.

The printDouble function works with any type (of data) that can be printed:

```
printDouble (Ham)
```

```
Ham Ham
```

printDouble (5)

5 5

printDouble (3.14159)

3.14159 3.14159

In the first call to the function, the argument is a string; in the second, it is an integer, and in the third, it is a floating-point number.

The same composition rules that apply to internal functions also apply to user-defined functions, so you can use any type of expression as an argument to printDouble.

printDouble (Ham * 4)

Ham Ham Ham Ham  Ham Ham Ham Ham
printDouble (math.cos (math.pi))
-1.0 -1.0

As usual, the expression is evaluated before executing the function, so printDouble returns Ham Ham Ham Ham Ham Ham Ham Ham Ham instead of 'Ham' * 4 'Ham' * 4.

We can also use a variable as an argument:

>>> latoya = 'Dafne, half laurel half nymph'

>>> printDouble (latoya) Dafne, half laurel half nymph. Dafne, half laurel half nymph.

Notice an important aspect in this case: the name of the variable that we pass as an argument (latoya) has nothing to do with the name of the parameter (step). It doesn't matter what the value was called in its original place (the place it was invoked from); here at printDouble we call everyone step.

# Modules and Packages

## Modules

A module allows you to logically organize your Python code. Grouping related code within a module makes the code easier to understand and use. A module is a Python object with arbitrary named attributes that you can bind and reference. Simply, a module is nothing but a file with a .py extension. A module can define functions, classes, and variables. It can also include executable code. Files are their physical counterpart: each Python file stored on disk is equivalent to one module. We are going to create our first module, then creating a small module.py file with the following content:

```
def my_function ():

    print "a function"

class MyClass:

    def __init __ (self):

        print "a class"

print "a module"
```

If we wanted to use the functionality defined in this module in our program, we would have to import it. To import a module, use the keyword import followed by the module name, which consists of the file name minus the extension. You can use any Python code file as a module by executing this statement in another Python code file. The import statement has the following syntax:

```
import you

import re, datetime
```

When the interpreter encounters an import statement, it imports the module if it is present in the search path. A search path is a list of directories that the interpreter searches before importing a module.

When creating our first module, it must be saved in the same directory as the program that imports it. But in some cases, we may need to import the modules from other directories. When importing a Python module, it goes through all the directories indicated in the PYTHONPATH environment variable in search of a file with the appropriate name. The value of the PYTHONPATH variable can be queried from Python using sys.path import sys sys.path

```
import sys

sys.path
```

In this way, for our module to be available to all the system programs, it would be enough to copy it to one of the directories indicated in PYTHONPATH.

The import clause also allows you to import multiple modules on the same line. In the following example, we can see how the modules of the Python os default distribution are imported with a single clause, which includes functionality related to the operating system; sys, with functionality related to the Python interpreter itself and time, in which functions are stored to manipulate dates and times:

```
import os, sys, time

print time.asctime ()
```

It is necessary to precede the name of the objects that we import from a module with the name of the module to which they belong, or what is the same, the namespace in which they are located. This allows us not to accidentally overwrite another object with the same name when importing another module. However, it is possible to use the from-import construction to save us having to indicate the name of the module before the object that interests us. In this way, the object or objects that we indicate are imported into the current namespace:

```
from time import asctime
```

print asctime ()

## Packages

A package is a folder that contains several modules. If modules are used to organize code, packages are used to organize modules. Packages are special types of modules (both are a type of module) that allow grouping related modules. While modules correspond on a physical level to files, packages are represented by directories. Using packages offers us several advantages. First of all, it allows us to unify different modules under the same package name, being able to create hierarchies of modules and sub-modules, or sub-packages. On the other hand, they allow us to easily distribute and handle our code as if they were installable Python libraries. In this way, they can be used as standard modules from the interpreter or scripts without previously loading them.

To make Python treat a directory as a package, you need to create a file __init__.py in that folder. This way, we get Python to understand that it is a package and not a simple folder. In this way, we can access some of the modules of the package. Like modules, import and from-import are also used to import packages to separate packages, sub-packages, and modules:

import package subpackage module

package.subpack.modulo.func ()

# Object-Oriented Programming

Object-Oriented Programming (OOP), as we have seen previously, is a programming paradigm. As such, it teaches us a method—proven and studied—which is based on the interactions of objects to solve t a computer system's needs.

### Elements and Characteristics of OOP

The elements of OOP can be understood as the materials we need to design and program a system, while the characteristics could be assumed as the tools that we have to build the system with those materials. Among the main elements of OOP, we can find classes, properties, methods, and objects.

Classes are the models on which our objects will be built. That is the generic area template from which to instantiate the objects, a template that is the one that defines what attributes and methods will have the objects of that class. In Python, a class is defined with the statement class followed by a generic name for the object:

```
class Object:
  pass
```

Properties, as we have seen before, are the intrinsic characteristics of the object. These are represented as variables, only they are called *properties* :

```
class Object ():
  color = ""
  size = ""
  aspect = ""
```

The methods are *functions,* technically called methods, that represent their own actions that the object (and not another) can

perform:

The methods are *functions*

```
class Object ():
  color = "gree"
  size = "big"
  aspect = "ugly"

def float (self):

pass
```

The classes by themselves are nothing more than models that help us create specific objects.

We can say that a class is an object's abstract reasoning, while the object is its materialization. Creating objects is called *instantiate a class,* and this instance consists of assigning the class as a value to a variable:

```
class Object ():

  color = "green"

  size = "big"

  aspect = "ugly"

  antennas = Antenna ()

  eyes = Eye ()

  hairs = Hair ()


def float (self):

print 12
```

```
et = Object ()

print et.color

print et.size

print et.aspect

et.color = "pink"

print et.color
```

## Heritage

Some objects share the same properties and methods as another object and also add new properties and methods. This is called inheritance: a class that inherits from another. However, the act of inheriting from a class is also often called "extending a class." It is worth clarifying that in Python when a class does not inherit from any other, it must be inherited from an object, which is the main Python class, which defines an object.

Suppose we want to model the musical instruments of a band, then we will have a guitar class, a drum class, a bass class, and so on. Each of these classes will have a series of attributes and methods, but it happens that, by the mere fact of being musical instruments, these classes will share many of their attributes and methods; an example would be the method play().

It is easier to create an Instrument object type with the common attributes and methods and tell the program that Guitar, Drums, and Bass are instrument types, making them inherit from Instrument. To indicate that one class inherits from another, place the name of the class it inherits from in parentheses after the name of the class:

```
class Instrument:

    def __init __ (self, price):

self.price = price
```

```
def play (self):
    print "We are playing music"
def break (self):
    print "You will pay that"
    print "Are", self.price, "$$$"
class Battery (Instrument):
    pass
class Guitar (Instrument):
    pass
```

As Drums and Guitar inherit from Instrument, they both have a method play() and a method break(), and are initialized by passing a price parameter. But what if we wanted to specify a new string_type parameter when creating a Guitar object? It would be enough to write a new method __init__for the Guitar class that would be played in place of the Instrument __init__. This is what is known as overriding methods. Now, it can happen in some cases that we need to overwrite a method of the parent class, but in that method, we want to execute the method of the parent class because our new method does not need more than to execute a couple of new extra instructions. In that case, we would use the SuperClass.method (self, args) syntax to call the method of the same name as the parent class. For example, to call the Instrument __init__method from Guitar, we would use Instrument __ init __ (self, price). Note that, in this case, it is necessary to specify the self parameter.

A class can inherit from multiple classes at the same time. For example, we could have a Crocodile class that inherits from the Terrestrial class, with methods like walk () and attributes like speed_walk, and of the Aquatic class, with methods like swim () and attributes like speed _swim. It is enough to enumerate the classes from which it is inherited, separating them by commas:

```
class Crocodile (Ground, Aquatic):
    pass
```

In the event that any of the parent classes had methods with the same name and number of parameters, the classes would overwrite the implementation of the methods of the classes further to their right in the definition. In the following example, as Terrestrial is more to the left, it would be the definition of displacement of this class that would prevail, and therefore if we call the displace method of an object of type Crocodile, what would be printed would be “The animal walks":

```
class Ground:
    def move (self):
        print "The animal walks"
class Aquatic:
    def move (self):
        print "The animal swims"
class Crocodile (Ground, Aquatic):
    pass
c = Crocodile ()
c.move ()
```

## Polymorphism

It means the ability to take more than one form. An operation can exhibit different behaviors in different instances. The behavior depends on the data types used in the operation. Polymorphism is widely used in the application of inheritance since an object of a derived class is at the same time an object of the parent class, so where an object of the parent class is required, one of the child class can also be used.

In Python, since it is not necessary to explicitly specify the type of the parameters that a function receives, functions are naturally polymorphic. A block of code will be polymorphic when within that code, calls are made to methods that can be redefined in different classes.

Using polymorphism, we can invoke the same method of different objects and obtain different results according to their class. This means that we can call a method exactly the same as another, and the interpreter will automatically detect which of them we refer to according to various parameters, for example, the type of data we pass as an argument when calling it, the class to which it belongs, or we can even specify which method we mean. Polymorphism is a matter of organization and good practice for the programmer who works with many objects and methods.

Allowing the developer not to write, think, and remember many different method names, but instead can call the appropriate object's method with the same name that it would call others.

Polymorphism is used very often, more than we are aware of. Remember that in Python, everything is an object, which makes it very likely that even without resorting to classes, we will use polymorphism. For example, only the print () function prints various types of objects without the need for us to specify anything, and this is also possible thanks to one of Python's properties called "dynamic typing."

## Method overload

In Python, method overloading as such does not exist. Those who come from other languages such as Java find some confusion since it is something very common in that language. In Python, overloading is absurd, although there are those who have written about it claiming that it is "good practice in any language."

Method overloading refers to the practice of having different methods with the same name in the same class. In this way, the interpreter or compiler will be able to differentiate them by the types of data that are sent as arguments in the parameters.

In Python, the dynamism that characterizes this programming language causes a conflict by not knowing what type of variables we are referring to. Although it is possible to emulate overloading, we will skip this practice since, although it works well, it is absurd and unnecessary. In Python, you should not have two methods with the

same name or call the same method with different types of parameters.

## Encapsulation

Encapsulation in programming is a concept related to object-oriented programming and refers to the hiding of a class's internal state from the outside. In other words, encapsulation consists of making the attributes or methods internal to a class not accessible or modified from the outside, but only the object itself can access them.

In Python, there are no access modifiers, and what is usually done is that access to a variable or function is determined by its name: if the name begins with two underscores (and does not also end with two underscores), it is of a private variable or function; otherwise it is public. Methods whose names begin and end with two underscores are special methods that Python calls automatically under certain circumstances:

```
Class Student ():
    Def_init_ (self.name = ""):
        Self.name = name
        Self._secret = "asdasd"
a1 = Student ("Josheph")
a1._secret
Traceback (most recent call last):
File “<stdin>”, line 1 in <module>
AttributeError: "Student" object has no attribute "_secret"
```

# Functional Programming

For a few years, it has become fashionable to work with functional programming as if the new generations had re-discovered the advantages over object-oriented programming or began to become aware of the limitations in their way of working.

The principles of functional programming are as follows:

- Using functions: As the name suggests, everything is built through functions. This way of working is not only simple, orderly, clear, easy to test, but it is also a practice that great military figures such as Julius Caesar and Napoleon have used. No, they did not use Python (at least there is no record of it), but they applied the concept of: "divide and conquer." And functional programming uses this strategy for just about everything.
- First-class features: Functions are treated as one more variable. They can even be returned.
- Pure functions: Fully predictive, the same input data will produce the same output data. You can override the input parameter without disturbing the flow of the program.
- Recursion: Functions can call themselves, simplifying tasks such as traversing data trees or managing controlled loops.
- Immutability: There are no variables, only constants. Personally understanding its potential and putting it into practice was like resetting my brain; I had to re-learn how to use a variable. Anecdote aside; Where does software usually fail? In the vast majority of cases, it comes from a variable that has been changed. This causes a block of code to be executed with conditions unforeseen. It is necessary to review each variable in different values until we find the culprit. I make a reflection: What if those variables were never modified? Or, being more practical, what if we create a new constant for each modification? What if... I tell you that at the performance level... it is more efficient? It is a very interesting concept to apply.
- Lazy evaluation (not strict): In functional programming, we can work with expressions that have not been evaluated, or in other words, we can have variables with operations whose result is not yet known. This is called loose screening. One side effect is increased performance, and another is that we can do crazy things like doing calculations with very complex operations or infinite lists without doing calculations. How is this possible? Because you work with mathematical expressions, the value is only calculated when you need it, for example, when performing a print.

Why use functional programming? Functional techniques facilitate the creation of concurrences, help us to reduce problems since the variables are constant and immutable so that we will not observe the programming error derived from the "Mutable global state." Likewise, testing is a faster process because by knowing what parameters to give to a function, we can know what results to expect. Functional programming is not exclusive. It is combined in an excellent way with imperative and object-oriented programming. Since version 3 of Python, this language has adopted native tools. Finally, we will obtain a compliant code that is easy to assimilate and read since it is more comfortable to understand a function than the structuring of an object.

Before starting with examples, it is worth knowing some of the modules that facilitate functional programming in Python. Among them, we can find the following:

- Intertools: This module comes already installed with the official distribution of python; it provides us with a large number of tools to facilitate the creation of iterators.
- Operator: We will also find this module already installed with Python, in which we will be able to find the main operators of Python turned into functions.
- Functools: Also already included within Python, this module helps us créate Higherorder functions, that is, functions that act on or return other functions.
- Fn: This module, created by Alexey Kachayev, gives Python additional "batteries" to make the functional style of programming much easier.
- Cytoolz: This module, created by Erik Welch, also provides several tools for Functional programming, specially oriented to data analysis operations.
- Macropy: This module, created by Li Haoyi brings to Python characteristics of purely functional languages, such as pattern matching, tail call optimization, Y case classes.

Python, without being a purely functional language, includes several characteristics taken from functional languages such as higher-order functions or lambda functions (anonymous functions).

## Higher-Order Functions

The concept of higher-order functions refers to the use of functions as if it were any value, making it possible to pass functions as parameters of other functions or to return functions as a return value. This is possible because, as we have already insisted on several occasions, in Python, everything are objects. And functions are no exception.

In the following lines, we can see an example of the above:

```
def greet (lang):
def greet_es ():
        print "Hello"
def greet_en ():
   print "Hi"
def greet_fr () :
    print "Salut"
lang_func = {“es”: greet_es,
        "En": greet_en,
        "Fr": greet_fr}
return lang_func [lang]
f = greet ("is")
F()
```

As we can see, the first thing we do in our little program is called the greet function with a parameter it is. In the greet function, several functions are defined: greet_es, greet_en, and greet_fr, and then a dictionary is created that has as keys text strings that identify each language, and functions as values. The return value of the function is one of these functions. The function to return is determined by the value of the parameter lang that was passed as an argument to say hello. Since the return value of hello is a function, as we have seen, this means that f is a variable that contains a function. We can then

call the function referred to by f in the way that we would call any other function, adding a few parentheses and, optionally, a series of parameters between the parentheses. This could be shortened since it is not necessary to store the function that is passed to us as a return value in a variable to be able to call it:

g reet (“en”) () Hi

greet (“fr”) () Sal ut

In this case, the first pair of parentheses indicates the parameters of the hello function, and the second pair, those of the function returned by hello.

## Using Map, Reduce, Filter, and Zip

One of the coolest things we can do with our higher-order functions is passing them as arguments to the map, filter, and reduce functions. These functions allow us to replace the typical loops of imperative languages with equivalent constructions.

## Map

The function map allows us to apply a function on each of the elements of a collection (lists, tuples, etc ...). We will use this function whenever we have the need to transform the value of one element into another. The structure of the function is as follows:

map (functon to apply, iterable object)

The function to be applied must return a new value. It is from these new values that we will obtain a new collection. Let's see an example:

```
def square (element = 0):
    return element * element

list = [1,2,3,4,5,6,7,8,9,10]

result = list (map (square, list))
    print (result)
```

As of version 3, the map function returns a map object, which we can easily convert to a list. In this case, as the function that we apply on the elements, we can replace it with a lambda function. The code could be as follows:

```
Result = list (map (lambda element: element * element, list))
```

### Filter

The filter function is perhaps one of the most used functions when working with collections. As its name indicates, this function allows us to filter the elements of the collection. The structure of the function is as follows:

```
Filter (function to apply, iterable object)
```

The *function to apply* will be applied to each of the elements of the collection. This function should always return a Boolean value. All those elements that result in *True a* fter applying this function, it will be the elements that pass the filter. From these elements, a new collection will be created. Let's see an example:

```
def greater_than_five (element):
  return element> 5
tuple = (5,2,6,7,8,10,77,55,2,1,30,4,2,3)
result = tuple (filter (greater_than_five, tuple))
result = len (result)
   print (result)
```

As of version 3, the filter function returns a filter object that we can easily convert to a tuple.

### Reduce

We will use the reduce function when we have a collection of elements, and we need to generate a single result. Reduce will allow

us to reduce the elements of the collection. We can see this function as an accumulator. The structure of the function is as follows:

reduce (function to apply, iterable object)

Here, the important thing is to detail the *function to apply* . This function must necessarily have two parameters. The first parameter will refer to the accumulator, a variable that will change its value for each of the elements in the collection. On the other hand, the second parameter will refer to each element of the collection. The function must return a new value. It will be this new value that will be assigned to the accumulator. This may sound confusing, but it will improve with a couple of examples. Let's start with an imperative approach:

```
list = [1,2,3,4]

accumulator = 0;

for element in list:

  accumulator + = element

    print (accumulator)
```

As we can see, to solve the problem, we had to declare a variable accumulator that starts with the value of 0. As we go through the list, the value of our variable increases. Its new value is the current value plus the value of the item in the list. So far, I don't think there is any doubt. Now let's look at the same example using the reduce function:

```
from functools import reduce

list = [1,2,3,4]

def function_accumulator (accumulator = 0, item = 0):

  return accumulator + element

  result = reduce (function_accumulator, list)

    print (result)
```

For each element of the collection, the function is executed, *function_accumulator* . The function returns the sum of the

parameters. This value is stored in our accumulator. At the end of the iteration of all the elements, reduce will return the value of the accumulator.

## Zip

The built-in function (i.e., it doesn't need to be imported) zip() takes as argument two or more iterable objects (ideally each one with the same number of elements) and returns a new iterable whose elements are tuples that contain one element from each of the original iterators.

countries =["China", "India", "United States", "Indonesia"]

population = [1391, 1364, 327, 264]

list (zip (countries, populations))

[("China", 1391), (“India”, 1364), (“United States”, 327), (“Indonesia”, 264)]

This function is especially useful in *for* loops to access the elements of two or more iterables simultaneously:

```
for country, population in zip (countries, population):
    print("{}: {} million inhabitants.". format (country, population))
```

China: 1391 million inhabitants.

India: 1364 million inhabitants.

United States: 327 million inhabitants.

Indonesia: 264 million inhabitants.

## Lambda Functions

The lambda operator is used to create anonymous functions online. As they are anonymous functions, that is, without a name, they cannot be referenced later. Lambda functions are constructed using the lambda operator, the function parameters separated by commas (attention, no parentheses), a colon (:), and the function code. This construction could have been useful in previous examples to reduce

code. The program we use to explain filter, for example, could be expressed like this:

```
l = [1,2,3]

12 = filter (lambda n: n% 2.0 == 0, 1)
```

Let's compare it to the previous version:

```
Def is_pair (n):

    Return (n% 2.0 == 0)

l = [1,2,3]

12 = filter (is_pair, 1)
```

Lambda functions are restricted by syntax to a single expression.

# Text Files

## About Files

Python makes working with files and text very easy. Let's start with the files.

Let's start with a short discussion about terminology. In a previous lesson, depending on the operating system of your computer, Mac or Windows , you saw how information is sent to the "exit command" window in your text editor by using the command print Python:

print (Hello world)

The Python programming language is of the object-oriented type. This means that it is built around a special type of entity, an object, which contains both data as well as a series of methods for accessing and altering the data. Once an object is created, it can interact with other objects.

In the example above, we saw one type of object, the string "Hello world." The string is the sequence of a series of characters enclosed in quotation marks. You can write a string in three different ways:

Message1 = 'Hello world'

Message2 = "Hello world"

Message3: "" "Hello

Hello

Hello world ”” ”

What is important here is to note that, as seen in the first two examples, you can use single or double quotes, but you should never mix the two types in the same string. In the third message, double quotation marks repeated three times indicate a string that spans more than one line.

Therefore the following messages contain errors:

Message1 = "Hello world"

Message2 ='Hello, world'

Message3 ='His name is John O'Connor'

Counts the number of single quotes in the message 3. For this to work correctly, we will have to save the apostrophe. Or rewrite the phrase as:

Message3 = 'His name is John O \ ´Connor'

Print is a command that prints objects in textual form. Combining the print command with a text string produces a statement.

You will use the print command in this way in cases where you want to generate information that needs to be manipulated immediately. Sometimes, however, you will create information that needs to be saved, sent to someone else, or used as input for further processing by another program or set of programs. In these cases, we will want to send information to files on the hard drive instead of sending it to the output command panel. Write the following program in your text editor and save it as file-output.py

f = open ('helloworld.txt', 'wb'

f.write ('Hello, world')

f. close ()

In Python, any line that starts with a pound sign or pound sign (#) is called *commentary* and is ignored by the Python interpreter. Comments are intended to allow programmers to communicate with each other (or to remind themselves of what the code is doing when they sit in front of it a few months later). In a broad sense, programs are written and formed in a way that makes it easier for programmers to work collectively. The code that is closest to the requirements of the machine is called a *low level* , while the code that is closest to the language of human beings is called *high level* . One of the benefits of using a programming language like Python is that it is of a higher level, which makes it easier for us to communicate with you (of course, at a certain cost in terms of computing efficiency).

In this show, f is an *object while open, write, and close are methods* . In other words, *open, write,* and *close* act on the object *f,* which, in this case, is defined as a .txt text file. This is probably a use of the term "method" that you might expect, and from time to time, you will find that words used in the context of programming have slightly (or completely) different meanings than everyday speech. In this case, remember that "method" means code snippets that perform actions. They run something on one thing and return a result. You can try to imagine this using some real-world referent, such as giving orders to your dog that has been trained previously. Your pet (the object) understands commands (i.e., it has "methods") like "bark," "sit," "lie down," and so on. We will discuss and learn how to use many other methods as we progress.

The name of a variable that we have chosen is f. We could have called him anything. In Python, variable names can be constructed with uppercase letters, lowercase letters, or numbers. But we cannot use the names of the language commands as variables. For example, if we try to name a variable "print," the program will not respond because that is a reserved word that is part of the programming language. Variable names in Python are also case-sensitive, which means that trap, Trap, or TRAP would be representations of different variables.

When you run the program we wrote, the method open tells your computer to produce a new text file called helloworld.txt in the same folder that we created the file-output.py program in. The *parameter w* indicates that we intend to write content to this new file using Python. Keep in mind that both the file name and the parameter are enclosed in single quotes, so you know that it will be data stored as strings. If you forget to include the quotes, the program will crash. In the next line, your program writes the message "Hello world" (which is another string) in the file and then closes it. Run Python. And although nothing will be written in the output command panel, you will see a status message that will say something like this on Mac:

'/ usr / bin / python file-output.py' returned 0.

While in Windows, you will see:

'C: \ Python27 \ Python.exe file-output.py' returned 0.

Since plain text files include minimal information, they tend to be small in volume, easy to exchange between different platforms (for example, from Windows to Linux or Mac or vice versa), and easy to send from one computer program to another. They can also be read in all text editors.

**Read From a Text File**

Python also has methods that allow us to get information from files. Write the following program in the text editor and save it as input-file.py . When you click "Run Python," the program will open the text file you just created, read the one-line text it contains, and print the information in the "output command" panel.

```
f = open ('helloworld.txt', 'r')

message = f.read ()

    print(message)

f. close ()
```

In this case, the parameter r is used to indicate that you are opening a file to read the information it contains. Parameters allow you to choose from a number of different options that a particular method allows. Going back to the pet example, the dog can be trained to bark once if it receives a beef-flavored treat and twice if it receives a chicken-flavored treat. The taste of the prize cookie is the parameter. Each method is different in terms of what parameters it will accept. For example, you can't ask the dog to sing an Italian opera—unless your dog is particularly talented. You can find the possibility of parameters for each particular method on the Python website, or you can even discover them yourself in any search engine by typing the specific method accompanied by the word "Python."

Read is another file method. The content of the file (the single-line message) is copied to the message, which is how we decide to call that text string, and the print command is used to send the content collected in the message to the output command panel.

## Append Text to an Existing Text File

A third option is to open an existing file and add more information to it. Note that if you open a file using open and you use the method write, the program will overwrite whatever the file contains. Of course, this is not a problem when creating a new file or when you want to overwrite the contents of an existing file, but it is totally undesirable when you are creating a long list of events or are compiling a large amount of data into an archive. So instead of write, we are going to use the method append, which is designated with a to.

Write the following program in the text editor and save it as file-apend.py. When you run it, this program will open the same helloworld.txt` text file that you created earlier and add a second “Hello world” to the file. The syntax '\ n' represents a new line of text in the file.

```
f = open ('helloworld.txt', 'a')
f.write ('\ n' + Hello world)
f. close ()
```

After you've run the program, go to the helloworld.txt file, and open it to see what happened. Close the text file and rerun the file-apend.py program as many times as you like. When you open the helloworld.txt file again, you will see that there will be a series of lines with the message “Hello World” repeated as many times as you run the program.

## Binary Files

Not all files are text files, and therefore not all files can be line processed. There are files in which each byte has a particular meaning, and it is necessary to manipulate them knowing the format in which the data is in order to process that information.

To open a file and handle it in the binary form, it is necessary to add a b to the mode parameter.

To process the file bytes instead of lines, use the function content = file.read (n) to read n bytes and file.write (content) to write content

to the current position of the file.

The b in the opening mode comes from binary, due to the binary numbering system, since in the computer processor, the information is handled only by zeros or ones (bits) that make up binary numbers.

Although it is not necessary for all systems (in general, the same system detects that it is a binary file without our asking), it is a good habit to use it, even though it serves mainly as documentation.

When handling a binary file, it is necessary to be able to know the current position in the file and to be able to modify it. To obtain the current position, use file.tell (), A that indicates the number of bytes since the beginning of the file.

To modify the current position, use file.seek (start, from), which allows moving a starting amount of bytes in the file, counting from the beginning of the file, from the current position, or from the end .

# Regular Expressions

One of the most frequent operations that you have come across is looking for a certain pattern/substring in a list, table, or text file.

This is not difficult if the pattern you are looking for is static, and you know it precisely. For example, if you want to find a certain name in a contact list, just use functions like find () , which are already included in Python.

But what happens when the substring you are trying to find has variants in its writing? For example, suppose that in a certain text, you want to find how many times the name “Händel” appears. Being a Germanic name, in our language, it can be written as "Händel," "Handel" or "Haendel". If you only use functions like find (), you will have to find each variant of the name separately. What if in a binary sequence you want to find all the subsequences of the form 010, 0110, 01110, 011110, and others? Now you can no longer use the find () method once for each case since there are infinite ones. Another way is needed.

This is where regular expressions come into the picture, a very powerful tool that makes it easy to find patterns in text. Regular expressions come from the world of theoretical mathematics, specifically the Theory of Formal Languages, but they are widely used in programming. Without going into mathematical definitions, a regular expression can be thought of as a word, made up of special characters, that serves to identify a set of other words.

I will illustrate it with an example. Let's go back to the beginning when we wanted to find the word Händel in a text. You could search for the three variants ("Handel," "Händel" and "Haendel") separately, or you could find another word that "encodes" these three variants. In this case, the word that is needed would be*"H (a | ä | ae) ndel"* . The vertical bar is a type of metacharacter used to separate the possible variants for the expression in parentheses. This word is a regular expression, as it identifies a set of other words.

## Metacharacters

A regular expression can contain metacharacters, which are symbols that have special meaning when placed inside a regular expression. In the previous section, you have seen the metacharacter |, which is equivalent to the Boolean expression or and is used to separate the alternatives of a word. However, there are many more metacharacters. Before we start with the Python examples, I'm going to show you some of the most common ones.

- The metacharacter '?' indicates "at most one match" of the character that comes immediately before. Thus, the expression "obscure" corresponds to the words "dark" and "obscure." The expression “(re)? Place” would correspond to “place” and “reposition.”
- The metacharacter '*' indicates "zero or more matches" of the character that comes immediately before. Thus, the expression "01 * 0" would correspond to the words 0, 010, 0110, 01110, 011110, etc.
- The metacharacter '+' works similar to the previous one, but indicates "at least one match". The expression "01 + 0" would now correspond to the words 010, 0110, 01110, 011110, etc.
- The metacharacter '{n}' indicates "exactly n matches" of the previous character. For example, the expression “ab {3} a” would correspond to the word “abba”.
- The metacharacter '{n, m}' indicates “between n and m matches”. If the second space is blank, it means "at least n matches". Therefore, the expression "01 {2,4} 0" would correspond to 0110, 01110, 011110, while the expression "01 {2,} 0" would accept the words 0110, 01110, 011110, 0111110, ...
- The metacharacter '.' is a wildcard that can be used in place of any other character. Thus, in a binary alphabet, the regular expression “01.0” would correspond to the words 0100 and 0110.

As you have seen in these examples, the parentheses () are also a metacharacter and are used to group terms and specify the order of operations. The expression “(01) * 0” is not the same as the expression “01 * 0”. The first corresponds to the words 0, 010, 01010, 0101010 and others while the second corresponds to the words 00, 010, 0110, 01110 ...

These are the most common metacharacters that you will find when working with regular expressions, although there are more. It is important to note that the syntax may vary a bit depending on the language and the context in which they are applied. Metacharacters can be combined to form more complex expressions. For example, the expression “fi. * (A | o)” would correspond to the words that begin with “fi” and end with the letter a / o: finite, philosophy, finalize, fixed, physical

**Sets**

A set is a set of characters enclosed in square brackets [] with a special meaning. Some of the most frequent are:

- [abc]- searches for a match with any of the characters in parentheses. The regular expression "[abc] aa" corresponds to the words aaa, baa, caa. It also works with numeric characters.

- [ak]- searches for a match with any of the alphabetic characters between the first (a) and the last (k). The regular expression "[be] a" corresponds to the words ba, ca, da, ea, while an expression like "[az] aa" would correspond to words that begin with any letter of the alphabet and end in "aa ”.

- [1-9]- is identical to the previous case, but with numeric characters.

- [^ abc]- matches all characters that are NOT inside the brackets. Thus, the regular expression “. * [^ A]” would correspond to any word that does NOT end with the letter “a”.

**Regex module**

To work with regular expressions in Python, you need the regex module. In the following examples, you will see some of its most basic methods.

To find a pattern in a string, we can use the method search ().

```
import re

text = "Lorem ipsum pain sit amet, consectetur adipiscing elit"

pattern = "Lorem"

x = re.search (pattern, text) #Searches the pattern inside the text

print (x.span ()) #Writes the initial position and the end of the occurrence
```

You can also use the method **match ()** , but this only returns a position if the occurrence is at the beginning of the text. This second code will give an error when trying to do the *print()* of the second match since the "ipsum" pattern is not at the beginning of the text, and therefore, the method *match()* bring back *None* :

```
import re

text = "Lorem ipsum pain sit amet, consectetur adipiscing elit door"

pattern1 = "Lorem"

pattern2 = "ipsum"

x = re.match (pattern1, text) # Searches the pattern inside the text

print (x.span ()) # Writes the initial position and the end of the occurrence

y = re.match (pattern2, text) # Return None

print (y.span ()) #ERROR!
```

So much *match()* as *search()* they only keep the first occurrence found. If you think there may be more than one, you can use the

function **find iter()** to search them all. This code looks for the pattern "pain" in a long text and writes its positions:

```
import re

text = "" "Lorem ipsum pain sit amet, consectetur adipiscing elit,

sed do eiusmod tempor incididunt ut labor et pain magna aliqua.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. "" "

pattern = "pain"

x = re.finditer (pattern, text) #Returns a vector with the occurrences positions

for i in x:

print (i.span ())
```

In the previous example, we have limited ourselves to searching for a static word in the text, but metacharacters can be used within the pattern to create regular expressions. I will put some examples.

In this first code, the regular expression "01 + 0" is used to find in a binary sequence all the sub-sequences that start and end with zero and only have ones in the middle: 010, 0110, 01110, etc.

```
import re

text = "01000100010011001"

pattern = "01 + 0"

x = re.finditer (pattern, text)

for i in x:

print (i.span ())
```

Of course, metacharacters can be combined to create more complex regular expressions. This code asks the user for a binary sequence and checks if it is capicua. The regular expression '(0. {3} 0) | (1. {3} 1)' accepts sequences that start and end with 0 or 1, with any three characters in between.

```
import re
text = raw_input ("Input a binary sequence of five figures: \ n")
pattern = "(0. {3} 0) | (1. {3} 1)"
valid = False
#The input is checked for validity
if len (text) == 5:
valid = True
for i in text:
if i! = '0' and i! = '1':
valid = False
break
#If the input is valid it is checked if it is capicua
if valid:
x = re.search (pattern, text)
if (x! = None):
print ("Is capicua!")
else:
print ("It is not capicua!")
#If the input is not valid and an error message appears:
else:
print ("ERROR: text is not valid")
```

This code uses sets to analyze a sequence of three-digit numbers and sticks with those that are odd. Here the method is used *in all()* that, unlike *find iter(),* returns a vector with the substrings of the occurrences instead of the positions.

```
import re
```

```
text = "551 889 302 105 012 817 894 206"

pattern = "[0-9] {2} [13579]"

x = re.findall (pattern, text) #Returns a vector with the substrings of the occurrences

for i in x:

print (i)
```

What does the regular expression “[0-9] {2} [13579]” that I just used mean? We know that a number will be odd if its last digit is odd. So I'm looking for numbers whose first two digits are between 0 and 9 and whose last digit is odd.

In this other example, the set [^] is used to discard all numbers that contain the digit '1'. The "space" character is also discarded to separate each of the numbers:

```
import re

text = "551 889 302 105 012 817 894 206"

pattern = "[^ 1] {3}"

x = re.findall (pattern, text) # Returns a vector with the substrings of the occurrences

for i in x:

print (i)
```

# Databases

A database is a file that is organized to store data. Most databases are organized like a dictionary in that they are mapped from keys to values. The biggest difference is that the database is on disk (or other permanent storage), so it persists after the program ends. Because a database is stored in permanent storage, it can store much more data than a dictionary, which is limited to the size of memory on the computer.

Like a dictionary, database software is designed to keep data entry and access very fast, even for large amounts of data. The database software maintains its performance by creating indexes as data is added to allow the computer to quickly jump to a particular entry.

### Database Peculiarities

However, the types of data and the way they are stored can differ greatly depending on the context exactly, and that is why, over time, a number of different models have been developed to manage databases.

- Hierarchical: The data that is organized in the form of an inverted tree uses a model where a parent node of information can have several children.
- Network: An improvement of the hierarchical model that allows a child to have multiple parents.
- ransactional: Whose sole purpose is the sending and receiving of data at high speeds. These bases are very rare.
- Relational: This is the model used today to represent real problems and manage data dynamically. It's the one we're going to focus on, but there are others.
- Documentaries: They allow us to save full text, and in general, to carry out more powerful searches. They serve to store large volumes of historical background information. Together with the relational ones, they are the most used in web development.
- Object-oriented: This model is quite recent and typical of object-oriented computer models, where it is about storing complete objects in the database. It is possible that it will take on more importance in the future.
- Deductives: They are databases that allow deductions. They are mainly based on rules and facts that are stored in the database, so they are somewhat complex.

Relational databases are widely used today because it is easy to represent and manage realworld problems. They are based on the idea of creating relationships between data sets, in which each relationship is also a table. Each table consists of records, made up of rows and columns, also known as tuples and fields. Obviously, within relational databases, there are many DBMS. Most are also compatible with Python. Some are paid, others free, some are simple, and others are very advanced. Let's do a review:

- SQL Server: It is a relational model database management system developed by Microsoft and only available for Windows systems. It is proprietary and a direct competitor of Oracle, MySQL, and PostgreSQL.
- Oracle: This database is a proprietary object-relational type database management system, developed by Oracle Corporation, considered one of the most complete systems. Its dominance in the enterprise server market was almost total until the appearance of the competition. It is multiplatform, and also the latest versions of Oracle have been certified to work under GNU / Linux.
- MySQL: It is a relational database management system developed under dual GPL / Commercial license by Oracle Corporation and is considered the most popular open-source database in the world, and one of the most popular in general together with Oracle and Microsoft SQL Server, especially for web development environments.
- PostgreSQL: It is a free object-oriented relational database management system. Like many other open-source projects, PostgreSQL development is not managed by one company or person but is led by a community of developers who work selflessly, altruistically, freely, or supported by commercial organizations.
- SQLite: It is a relational database management system contained in a small library written in C. It is a public domain project, and unlike the other systems that use the client-server architecture, its engine is not an independent process, but rather it links with the program by becoming an integral part of it. However, do not be fooled, because although it seems like the simple solution, SQLite in its third version allows databases of up to 2 Terabytes in size and many other functionalities. In short, its configuration is very simple, so simple that it does not exist, so it will not cause problems, and it is the best solution for this course.

As you can see, we find many Relational DBMS. In Python, each of them have free modules and connector programs to communicate the databases and the programming language. Even though they are different systems, the query language does not vary much.

Otherwise, it would be difficult to go from one system to another, and the DBMS could not compete with each other.

Apart from programming languages such as Python, focused on the creation of programs, DBMS implement their own syntax or language to make queries and modifications to their registers. The most used language in relational databases is SQL (Structured Query Language), and it is necessary to learn it if we want to use this type of database in our programs. Obviously, this language covers a lot, so in this unit, we will only see some basic queries to use in conjunction with SQLite in our Python scripts. While this chapter will focus on using Python to work with data in SQLite database files, many operations can be more conveniently performed using the software called Database Browser for SQLite, which is freely available at:
http://sqlitebrowser.org/

With the browser, you can easily create tables, insert data, edit data, or run simple SQL queries. In a sense, the database browser is similar to a text editor when working with text files. When you want to perform one or very few operations on a text file, you can open it in a text editor and make the changes you want. When you have many changes to make to, you will often write a simple Python program. You will find the same pattern when working with databases. You will do simple operations in the database manager, and more complex operations will be performed more conveniently in Python.

## Creating a Database Table

When you first look at a database, it looks like a spreadsheet with multiple sheets. The main data structures in a database are tables, rows, and columns. In technical descriptions of relational databases, the concepts of the table, row, and column are more formally called relation, tuple, and attribute, respectively. We will use less formal terms in this chapter.

Databases require a more defined structure than Python lists or dictionaries. When we create a table in the database, we must tell the database in advance the names of each of the columns in the

table and the type of data we plan to store in each of them. When the database software knows the type of data in each column, you can choose the most efficient way to store and search the data based on the type of data. You can see the various data types supported by SQLite at the following URL: http://www.sqlite.org/datatypes.html

Defining the structure of your data in advance may seem inconvenient at first, but the payoff is quick access to your data, even when the database contains large amounts of data. The code to create a database file and a table called Tracks with two columns in the database is the following:

```
import SQLite 3

con = sqlite3.connect ('music.sqlite')

cur = conn.cursor ()

cur.execute ('DROP TABLE IF EXISTS Tracks')

cur.execute ('CREATE TABLE Tracks (title TEXT, plays INTEGER'))

conn.close ()

#Code: http://www.py4e.com/code3/db1.py

$ Or select Download from this trinket's left-hand menu
```

The operation connect makes a "connection" to the database stored in the filemusic.sqlite3 in the current directory. If the file does not exist, it will be created. The reason this is called a "connection" is that sometimes the database is stored on a "database server" separate from the server on which we run our application. In our simple examples, the database will just be a local file in the same directory as the Python code we are running. A cursor is like a filehandle that we can use to perform operations on the data stored in the database. Call to cursor() is conceptually very similar to calling open () when dealing with text files.

Once we have the cursor, we can start executing commands on the content of the database using the method execute(). The database commands are expressed in a special language that has been standardized across many database vendors to allow us to learn a

single database language. The database language is called Structured Query Language or SQL.

http://en.wikipedia.org/wiki/SQL

In our example, we are running two SQL commands against our database. As a convention, we will display SQL keywords in uppercase, and the parts of the command that we are adding (such as table and column names) will be displayed in lowercase. The first SQL command drops the table Tracks from the database if it exists. This pattern is simply to allow us to run the same program to create the table Tracks over and over without causing an error. Note that the command DROP TABLE removes the table and all its contents from the database (that is, there is no "undo").

```
Cur.execute ('DROP TABLE IF EXISTS Tracks')
```

The second command creates a table called tracks with a column of text called title and an integer

```
cur.execute ('CREATE TABLE Tracks (title TEXT, plays INTEGER'))
```

Now that we have created a table called tracks, we can put some data in that table using SQL INSERT operation. Again, we start by making a connection to the database and getting the cursor. Then we can execute SQL commands using the cursor.
The SQL INSERT command indicates which table we are using and then defines a new row by listing the fields we want to include (title, plays) followed by the VALUES we want to place in the new row. We specify the values as question marks (??) to indicate that the actual values are passed as a tuple ('My Way' 15) as the second parameter to the call execute().

```
import sqlite3

conn = sqlite3.connect ('music.sqlite')

cur = conn.cursor ()

cur.execute ('DROP TABLE IF EXISTS Tracks')

cur.execute ('CREATE TABLE Tracks (title TEXT, plays INTEGER)')
```

```
cur.execute ('INSERT INTO Tracks (title, plays) VALUES (?,?)',
  ('Thunderstruck', 20))
cur.execute ('INSERT INTO Tracks (title, plays) VALUES (?,?)',
  ('My Way', 15))
conn.commit ()
print ('Tracks:')
cur.execute ('SELECT title, plays FROM Tracks')
for row in cur:
print (row)
```

First, we insert two rows into our table and use commit() to force the data to be written to the database file.

| Tracks | |
|---|---|
| title | plays |
| Thunderstruck | twenty |
| My way | fifteen |

Then we use the command SELECT to retrieve the rows we just inserted from the table. In command SELECT, we indicate which columns we would like to extract (title, we play), and we indicate from which table we want to retrieve the data. After executing the statement SELECT, the cursor is something we can loop through in an instruction for. For efficiency, the cursor does not read all the data from the database when we execute the statement SELECT. Instead, the data is read as we walk through the rows in the declaration for. The output of the program is as follows:

```
Tracks:
('Thunderstruck', 20)
```

('My Way', 15)

Our loop for finds two rows, and each row is a Python tuple with the first value as title and the second value as the number of plays.

At the end of the program, we run an SQL command to 'CLEAR' the rows we just created so that we can run the program over and over again. The command DELETE shows the use of a clause WHERE which allows us to express a selection criteria so that we can ask the database to apply the command only to the rows that match the criteria. In this example, the criteria now apply to all rows, so we empty the table so we can run the program repeatedly. After the DELETE, we also call commit() to force the data to be removed from the database.

## SQL Basics

So far, we have been using the structured query language in our Python examples and have covered many of the basics of SQL commands. In this section, we discuss the SQL language in particular and provide an overview of SQL syntax. Since there are so many different database vendors, Structured Query Language (SQL) was standardized so that we could easily communicate with multi-vendor database systems. A relational database is made up of tables, rows, and columns. Columns generally have a type such as text, numeric, or date data. When we create a table, we indicate the names and types of the columns:

CREATE TABLE Tracks (title TEXT, plays INTEGER)

To insert a row in a table, we use the SQL commandINSERT:

INSERT INTO Tracks (title, plays) VALUES ('My Way', 15)

When using the declaration insert, specify the table name, then a list of the fields/columns you would like to set in the new row, and then the keyword VALUES and a list of corresponding values for each of the fields. The SQL command SELECT used to retrieve rows and columns from a database. The declaration SELECT allows you to specify which columns you want to retrieve, as well as a clause WHERE to select which rows you want to see. It also allows an optional clause order BY to control the sorting of the returned rows.

SELECT * FROM Tracks WHERE title = 'My Way'

Using * indicates that you want the database to return all columns for each row that matches the clause WHERE. Note that unlike Python, in an SQL clause WHERE, we use a single equals sign to indicate a test of equality instead of a double equals sign. Other logical operations allowed in a clause WHERE include <,>, <=,> =,! =, as well as AND / OR and parentheses to build your logical expressions. You can request that the returned rows be ordered by one of the fields as follows:

SELECT title, plays FROM Tracks ORDER BY title

To delete a row, you need a clause WHERE in an SQL statement DELETE. The clause WHERE determines which rows are to be removed:

DELETE FROM Tracks WHERE title = 'My Way'

It's possible UPDATE a column or columns within one or more rows in a table using SQL statement UPDATE as follows:

UPDATE Tracks SET plays = 16 WHERE title = 'My Way'

The declaration UPDATE specifies a table and then a list of fields and values to change after the keyword SET and then an optional clause WHERE to select the rows to update. Single instruction UPDATE will change all rows that match the clause WHERE. If no clause is specified WHERE, perform the UPDATE in all rows of the table. These four basic SQL commands (INSERT, SELECT, UPDATE, and DELETE) enable the four basic operations required to create and maintain data.

# Error Handling in Python

In software development, different types of errors can occur. These could be syntax errors, logic errors, or runtime errors. Syntax errors are more likely to occur during the initial development phase and are the result of incorrect syntax. Syntax errors can easily be caught when compiling the program for execution. Logical errors, on the other hand, are the result of incorrect logic implementation. An example would be a program accessing an unordered list assuming it was ordered. Logical errors are the hardest to track down. Runtime errors are the most interesting errors that occur if we don't consider all possible cases. An example would be trying to access a non-existent file.

In this section, we will learn how to handle errors in Python and how to log errors for a better understanding of what happened within the application.

### Syntax errors

Syntax errors, also known as misinterpretations, are perhaps the most common type of complaint you get when you're still learning Python:

```
while True print 'Hello world'
Traceback (most recent call last):
...
  while True print 'Hello world'
^
SyntaxError: invalid syntax
```

The interpreter repeats the faulty line and displays a small 'arrow' pointing to the first place where the error was detected. This is caused by (or at least detected in) the symbol that precedes the

arrow: in the example, the error is detected in the print statement since there is a missing colon (':') before it. The file name and line number are displayed, so you know where to look if the input is coming from a program.

**Exceptions**

Even if the statement or expression is syntactically correct, it can generate an error when you try to execute it. Errors caught during execution are called exceptions, and they are not unconditionally fatal: you will soon learn how to handle them in Python programs. Let's start with a simple program to add two numbers in Python. Our program takes two parameters as input and prints the sum. Here is a Python program for adding two numbers:

```
def addnumbers (a, b):

print a + b

addNumbers (5, 10)
```

When writing the above program, we didn't really consider the fact that something could go wrong. What if one of the parameters passed is not a number?

```
addNumbers ('', 10)
```

We have not handled that case; therefore, our program will break with the following error message:

```
Traceback (most recent call last):

File "addNumber.py", line 4, in <module>

addNumbers ('', 10)

File "addNumber.py", line 2, in addNumbers

print a + b

TypeError: cannot concatenate 'str' and 'int' objects
```

We can handle the above question by checking if the parameters passed are integers. But that does not solve the problem. What if the code breaks for some other reason and causes the program to crash?

Working with a program that breaks when encountering an error is not a good sight. Even if an unknown error has occurred, the code must be robust enough to handle the break gracefully and let the user know that something is wrong.

In Python, we use the declarations tryYexcept to handle exceptions. When code breaks, an exception is thrown without crashing the program. We are going to modify the program that adds numbers to include the declarations tryYexcept.

```
def addNumbers (a, b):

try:

return a + b

except exception as e:

return 'Error occurred:' + str (e)

print addNumbers ('', 10)
```

Python would process all the code inside the statements try and except. When it encounters an error, control is passed to the block except, skipping the code in the middle. As seen in the above code, we have moved our code within a statement try and except. You try to run the program, and it should throw an error message instead of the program crashing. Also, the reason for the exception is returned as an exception message. The above method handles unexpected exceptions. Let's take a look at how to handle an expected exception. It assumes that we are trying to read a file with our Python program, but the file does not exist. In this case, we will handle the exception and let the user know that the file does not exist when it occurs. Take a look at the file reading code:

```
try:

try:

with open ('fname') as f:

content = f.readlines ()

except IOError as e:
```

print str (e)

except exception as e:

print str (e)

In the above code, we have handled the file reading inside an exception handler IOError. If the code breaks due to the unavailability of the file name, the error would be handled within the controller IO Error. Similar to the exception IO Error, there are many more standard exceptions like Arithmetic, Overflow Error, and Import Error, to name a few.

We can handle multiple exceptions at once by putting together the standard exceptions as shown:

try:

with open ('fname') as f:

content = f.readlines ()

printb

except (IOError, NameError) as e:

print str (e)

The above code would show the exceptions IOError and NameError when the program runs.

Suppose we are using certain resources in our Python program. During the execution of the program, an error was encountered, and it was only half executed. In this case, the resource will be kept unnecessarily. We can clean up such resources using the clause finally. Take a look at the following code:

try:

filePointer = open ('fname', 'r') try: content = filePointer.readline () finally:

filePointer.close ()

except IOError as e:

```
print str (e)
```

If, during the execution of the above code, an exception occurs while reading the file, the file Pointer would be closed on the block finally.

## Records

When something goes wrong within an application, it is easier to debug if we know the source of the error. When an exception occurs, we can record the information necessary to locate the problem. Python provides a simple and powerful registry library. Let's take a look at how to use registers in Python.

```
import logging

# initialize the log settings

logging.basicConfig (filename = 'app.log', level = logging.INFO)

try:
logging.info ('Trying to open the file')
filePointer = open ('appFile', 'r')
try:
logging.info ('Trying to read the file content')
content = filePointer.readline ()
finally:
filePointer.close ()

except IOError as e:

logging.error ('Error occurred' + str (e))
```

As seen in the above code, we first have to import the Python log library and then initialize the logger with the log file name and log level. There are five log levels: DEBUG, INFO, WARNING, ERROR, and CRITICAL. Here we have to set the log level to INFO; therefore, INFO and the previous logs will be logged.

In the above code, we had a program file, so it was easier to figure out where the error had occurred. But what do we do when it comes to multiple program files? In such a case, getting the error stack helps in finding the source of the error. The exception stack trace may have been logged as shown:

```
import logging
# initialize the log settings
logging.basicConfig (filename = 'app.log', level = logging.INFO)
try:
filePointer = open ('appFile', 'r')
try:
content = filePointer.readline ()
finally:
filePointer.close ()
except IOError as e:
logging.exception (str (e))
```

If you try to run the above program when an exception occurs, the following error is logged in the log file:

```
ERROR: root: [Errno 2] No such file or directory: 'appFile'
Traceback (most recent call last):
File "readFile.py", line 7, in <module>
filePointer = open ('appFile', 'r')
IOError: [Errno 2] No such file or directory: 'appFile'
```

# Python Web Development

## Contributions of Python to Web Development

Python is a programming language that can also be used to create personal and business web pages. Knowing how to program websites and applications is synonymous with success and a job gap. The future is in the network, and professionals capable of placing companies in this new medium of commerce are needed. It is because of the above that many programmers have chosen to specialize in this category.

Python webs are chosen every day by high-level companies, a sign of the quality and good usability offered by webs created with this code. Thus, programming in Python is one of the skills most sought after by the headhunters of companies such as Netflix, Spotify, and Pixar. Choosing Python web ensures you are not pigeonholed into a single type of web development. It is an opensource whose creation possibilities are endless, as well as its direct applications. With a good study on Python, you will learn to make websites from scratch with all its functionalities, from ecommerce to a platform with products on consumer demand like Netflix.

Python code has numerous advantages at the time of sitting down to create webs. We detail some of them below:

- It is an open-source language, which means that it is free software. You can use it for free at all times.
- It is multiplatform, valid for all types of systems and devices.
- It is understandable to all programmers. The way of writing Python code is neat and clean, so anyone will know how to understand and work on your created structure.
- It has a flexible style. With Python web, you can create a multitude of elements, from lists to more complex functions.
- It has a healthy programming style. Python web is designed to work according to specific rules so that everyone follows the same guidelines.

## Web Frameworks for Python

A web framework is a set of components that help you develop websites easier and faster. When you build a website, you always need a set of similar components: a way to handle user authentication (register, login, logout), an administration panel for your website, forms, a way to upload files, etc. Luckily for us, other developers realized long ago that they always faced the same problems when building websites, so they came together and built frameworks (such as Django) with out-of-the-box components. The frameworks serve so that we do not have to reinvent the wheel each time and that we can move faster when building a new site.

To understand what frameworks are really for, we need to look at how servers work. The first thing is that the server needs to know that you want a web page to serve you. Imagine a mailbox (port) in which someone is constantly looking for incoming letters (requests). This is what a web server does. The web server reads the letter and sends a response to the web page. But to send something, we have to have some content. And frameworks help us create that content. When a request arrives at a web server, it is passed to the framework, which tries to find out what is actually requested. Take a website address first and try to figure out what to do with it. This part is done by the URL resolver of our framework (note that a website address is called URL - Uniform Resource Locator, so the name URL resolver makes sense). This one isn't very smart - it takes a list of patterns and tries to match the URL. The framework checks

the patterns from top to bottom, and if something matches, it passes the request to the associated function (which is called view). Imagine a postman carrying a letter. He walks down the street and checks each house number with the one on the letter. If it matches, he leaves the letter there.

All the cool things are done in the view function: we can look at a database to find some information. Maybe the user asked to change something in the data, like a letter saying, "Please change my job description." The view can check if you have permission to do so, update your job description, and return a message: "Done!" The view then generates a response, and Django can send it to the user's browser. This description is a bit simplistic, but at the moment, you don't need to know all the technical details. Just having a general idea is more than enough.

There are quite a few Python web frameworks out there, but here are some of the best:

- Django: Django is a high-level web framework that enables rapid development of safe and maintainable websites. Developed by seasoned programmers, Django takes care of much of the complications of web development, so you can focus on writing your application without reinventing the wheel. It's free and open-source, has a thriving and active community, great documentation, and many free and paid support options.
- Grok: Grok, defined by its authors, is a framework for web applications made for Python developers. It is aimed at beginners and experts. Grok is intended for agile development. This framework comes from the well-known Zope, a complete dean in the world of application servers in web environments, of which the Zope Toolkit is used, a set of objectoriented libraries specially oriented to the reuse of components in web environments.
- Webpy: This is the software written by Aaron Swartz for creating dynamic websites with Python. It was officially posted while I was working on reddit.com (which was later rewritten using other tools). “Simple as powerful”, “It is the anti-framework framework. web.py doesn't get in your way"; some of the characteristics and opinions for which it is made known on the official website webpy.org. It is in the public domain, so it can be applied for the development of any type of project.
- Turbogears: TurboGears is a Python web application framework that consists of various WSGI components such as WebOb, SQLAlchemy, Genshi, and Repoze. TurboGears is designed around the model-view-controller (MVC) architecture, very similar to Struts or Ruby on Rails, designed to make rapid web application development in Python easier and easier to maintain. Since version 2.3, the framework has also provided a "minimal mode" that allows it to act as a microframe for use in environments where the full stack is not required or desired.

## Using Django

We will focus on introducing ourselves to web development in Python in the Django framework because this helps you write software that is:

- Full: Django follows the "Batteries Included" philosophy and provides almost everything developers would like it to have "out of the box". All you need is part of a single "product," everything works perfectly, follows consistent design principles, and has a wide and updated documentation.
- Versatile: Django can be (and has been) used to build almost any type of website - from content management systems and wikis to social media and news sites. It can work with any framework on the client-side, and it can return content in almost any format (including HTML, RSS feeds, JSON, XML, etc.). The site you are currently reading is based on Django! While it offers options for almost any functionality you want (different database engines, template engines, etc.), it can be extended to use other components if necessary.
- Insurance: Django helps developers avoid several common security mistakes by providing a framework that is designed to "do the right thing" to protect the website automatically. For example, Django provides a secure way to manage user accounts and passwords, thus avoiding common mistakes such as placing session information in cookies where it is vulnerable (instead, cookies only contain a key and the data is stored in the database data) or passwords are stored directly in a password hash. Django allows protection against some vulnerabilities by default, including SQL injection, cross-site scripting, cross-site request spoofing, and clickjacking.
- Scalable: Django uses a component based on the architecture "shared-nothing" (Each part of the architecture is independent of the others, and therefore can be replaced or changed if necessary). Taking into account a clear separation between the different parts means that you can scale to increase traffic by adding hardware at any level: cache servers, database servers, or application servers. Some of the busiest sites have scaled Django to meet their demands (for example, Instagram and Disqus, to name just two).
- Maintainable: Django code is written using design principles and patterns to encourage the creation of maintainable and reusable code. It uses the "Don't Repeat Yourself" (DRY) principle so there is no unnecessary duplication, reducing the amount of code. Django also encourages grouping related

amount of code. Django also encourages grouping related functionality into reusable "applications" and, at a lower level, groups related code into modules (following the pattern model View Controller (MVC)) .

- Portable: Django is written in Python, which runs on many platforms. This means you are not tied to any particular platform and can run your applications on Linux, Windows, and Mac OS X distributions. Additionally, Django is supported by many web-hosting providers, and they often provide specific infrastructure and documentation for hosting Django sites.

## Django Origins

Django was initially developed between 2003 and 2005 by a team that was responsible for creating and maintaining newspaper websites. After creating several sites, the team began to consider and reuse many common design codes and patterns. This common code became a generic web framework, which was open source, known as the "Django" project in July 2005.

Django continues to grow and improve from its first milestone, the release of version (1.0) in 2008, to the recent release of version 1.11 (2017). Each release has added new functionality and bug fixes, ranging from support for new database types, template engines, caching to the addition of generic functions, and display classes (which reduce the amount of code developers have to write).

Django is now a thriving collaborative open source project, with thousands of users and contributors. While it still has some features that reflect its origin, Django has evolved into a versatile framework that is capable of developing any type of website.

## Django Possibilities

The preceding sections show the main features that in almost all web applications: URL mapping, views, models, and templates. Just a few other things that Django provides, including:

- Forms: HTML forms are used to collect user data for processing on the server. Django simplifies the creation, validation, and processing of forms.
- User authentication and permissions: Django includes a robust authentication and permission system that has been built with security in mind.
- Search: Dynamic content creation is much more computationally intensive (and slow) than a static content service. Django provides flexible caching so that you can store all or part of a rendered page so that it is not re-rendered only when necessary.
- Administration site: The Django admin site is included by default when you create an app using the basic skeleton. This makes it trivially easy to provide an administration page for administrators to create, edit, and view any of their site's data models.
- Data serialization: Django makes it easy to serialize and serve your data as XML or JSON. This can be useful when creating a web service (a website that only serves data to be consumed by other applications or sites and does not display anything on its own) or when creating a website where the client-side code handles all the rendering of the data.

## Virtual Environment

Django makes it easy to set up your computer so you can start developing web applications. This section explains what you get out of the development environment and provides an overview of some of your startup and configuration options. It explains the recommended method of installing the Django development environment on Mac OS X and Windows and how you can test it.

The development environment is an installation of Django on your local computer that you can use to develop and test Django apps before deploying them to the production environment. The main tools that Django provides are a set of Python scripts for creating and working with Django projects, along with a simple development web server that you can use to test locally (i.e., on your computer,

not on a server. external web) Django web applications with your computer's web browser.

When you install Python3, you get a single global environment that is shared with all Python3 code, while you can install the packages you like in your environment. You can only install one particular version of each package at a time. If you install Django in the default / global environment, you will only be able to target a single version of Django on the computer. This can be a problem if you want to create new sites using the latest version of Django but keep websites that depend on older versions. Experienced Python / Django developers run Python applications within separate Python virtual environments. This enables multiple different Django environments on the same computer. The same Django development team recommends that you use virtual Python environments!

The libraries that we will use to create our virtual environments are in virtualenvwrapper (Mac OS X) and virtualenvwrapper-win (Windows), which in turn use the tool virtualenv. The toolswrapper creates a consistent interface for managing interfaces on all platforms.

**Mac OS X**

Install virtualenvwrapper (virtualenv*i* ncluded in the package) using pip3 shown in the following.
Sudo pip3 install virtualenvwrapper

Then add the following lines to the end of the start file of your shell. The startup file is called differently .bash_profile and it's hidden in your home directory.

export WORKON_HOME = $ HOME / .virtualenvs

export VIRTUALENVWRAPPER_PYTHON = / usr / bin / python3

export PROJECT_HOME = $ HOME / Devel

source /usr/local/bin/virtualenvwrapper.sh

Then reload the startup file by making the following call in the terminal:

source ~ / .bash_profile

At this point, you should see a handful of scripts starting to run. You should now be able to create a new virtual environment with the command mkvirtualenv.

### Windows

Installing virtualenvwrapper-win is even simpler than starting virtualenvwrapper because you don't need to configure where the tool stores the environment information (there is a default value). All you need to do is run the following command in the online command console:
pip3 install virtualenvwrapper-win

And then you can create a new virtual environment with mkvirtualenv

### Taking Advantage of Virtual Environments

Once you have installed virtualenvwrapper or virtualenvwrapper-win, working with virtual environments is similar on all platforms. Now you can create a new virtual environment with the command mkvirtualenv. As this command is executed, you will see that the environment starts up (what you will see is slightly platform-specific). When the command is completed, the new virtual environment will be active - you can check it because the beginning of the prompt will be the name of the environment in parentheses (as shown below).

$ mkvirtualenv my_django_environment

Running virtualenv with interpreter / usr / bin / python3

...

virtualenvwrapper.user_scripts creating

/home/ubuntu/.virtualenvs/t_env7/bin/get_env_details

(my_django_environment) ubuntu @ ubuntu : ~ $

Once you are inside the virtual environment, you can install Django and start development. There are just a few other useful commands

you should know about (there are more in the tool's documentation, but these are the ones you will use regularly:

- deactivate - Exit the current Python virtual environment
- workon- List the virtual environments available
- workon name_of_environment- Activate the specified Python virtual environment
- rmvirtualenv name_of_environment- Delete the specified environment.

## Django Installation

Once you have created the virtual environment and made the callworkonto get into it, you can use pip3 to install Django.

pip3 install Django

You can check that Django is installed by running the following command (this checks that Python can find the Django module):

# Mac OS X

python3 -m Django --version

1.11.7

# Windows

py -3 -m Django --version

1.11.7

The above test works, but it's not very fun. A more interesting check is to create a project skeleton and see if it works. To do this, navigate in your command console/terminal to where you want to store your Django applications. Create a folder for your site verification and navigate to it.

mkdir django_test

cd django_test

You can then create a new site skeleton called "mytestsite" using the tool Django-admin as it's shown in the following. After creating the site, you can navigate to the folder where you will find the main script for project management, called manage.py.

Django-admin startproject mytestsite

cd mytestsite

We can start the development web server from this folder using manage.py and the command runserver, as shown.

$ python3 manage.py runserver

Performing system checks ...

System check identified no issues (0 silenced).

You have 13 unapplied migration (s). Your project may not work properly until you apply the migrations for app (s): admin, auth, contenttypes, sessions.

Run 'python manage.py migrate' to apply them.

September 19, 2016 - 23:31:14

Django version 1.10.1, using settings 'mysite.settings'

Starting development server at http://127.0.0.1:8000/

Quit the server with CONTROL-C.

Once you have the server running, you can view the site by navigating to the following URL in your local web browser: http://127.0.0.1:8000/.

## Website Project: Blog

We will build a blog system today to learn the basics. First, we will create a Django project.

cd ~ / Documents / Projects

Django-admin.py startproject FirstBlog

cd FirstBlog

ls

What do these files do?

- _init_.py tells Python that this folder is a Python package. We learned about this in the third lesson; it allows Python to import all the scripts in the folder as modules.
- manage.py is not really part of your website; it is a utility script that you run from the command line. It contains an array of functions to manage your site.
- settings.py contains your website settings. Django doesn't use XML files for configuration; everything is Python. This file is simply a number of variables that define your site settings.
- URLs.py is the file that assigns the URLs to the pages. For example, you could map yourwebsite.com/about to an "About Us" page.

## Applications

However, none of these files create a functional website. For that, we need applications. Applications are where the code that makes the website work is written, but before we take a look at them, we need to get a little understanding of Django's design principles.

First, Django is an MVC framework, which stands for Model View Controller. Django refers to itself as an MTV framework, which stands for Model Template View. It is a slightly different approach than MVC, but fundamentally, they are quite similar. MVC is an architectural pattern that provides a method to structure your projects. Separates the code that is used to process data from the code that manages the user interface.

Second, Django subscribes to the DRY, or Don't Repeat Yourself philosophy, which means you should never write code that performs a certain task more than once. For example, on our Blog, if we wrote a function that picked a random article from the archive and

implemented this function on multiple pages, we would not re-code it every time it was needed. We code it once and then use it on every page.

So how does this relate to apps? Well, the applications allow you to write your website in a DRY style. Each project, like the one we have here, can contain multiple applications. Rather, each application can be part of multiple projects. Using the example from before, this means that if we created another site in the future that also needed a random page function, we wouldn't have to rewrite it. We could just import the application from this project. Because of this, it is important that each application has a different purpose. If you write all the functionality of your site inside one app, and then you need to use part of it again later, you have to import it all. If you were creating an e-commerce website, for example, you wouldn't want to import all the blog features. However, if you create an app for the shuffle function and an app for the blog publishing system, you can choose the bits you need.

This also means that within site, the code is well organized. If you want to modify a feature, you don't have to search a massive archive; instead, you can navigate to the corresponding app and change it without worrying about interfering with anything else.

python mangage.py startapp blog

cd blog

ls

Again, we have a file _init_.py to convert it into a package and three other files: models, tests, and views. We don't need to worry about testing for now, but the other two are important. Models and Views are the parts M and V by MVC.

When you want to access that data, go through these models by calling the method on them, rather than running raw queries. This is very useful because Django can use multiple database programs. We are going to use MySQL today because it is the most powerful, and it is what most hosts offer. However, if we need to switch to a different database in the future, all the code will still be valid! In other languages, if you wanted to switch to SQLite or something similar,

you would need to rewrite the code that accesses your database. In the views file, we write the code that generates the web pages. This ties all the other parts together. When typing a URL, it is sent through the scripturls that we saw before the script ofviews, which then obtains relevant data from the models, processes it, and passes it to a template, which is finally served as the page the user sees. We'll take a look at those templates shortly. They are the easiest part, mainly HTML.

For a blog, we will need a table of posts, with various fields for the title, body text, author, time of writing, etc. An actual blog demo.

```
from Django.db import models

class posts (models.Model):

author = models.CharField (max_length = 30)

title = models.CharField (max_length = 100)

bodytext = models.TextField ()

timestamp = models.DateTimeField ()
```

**MySQL**

These models are only a description. We need to make a real database of them. First, however, we need MySQL to run on our system. On a real webserver, this wouldn't be a problem as they usually have it pre-installed. Fortunately, with a package manager, it is easy to install. First, you need to install Homebrew and Easy installation.

```
brew install MySQL

easy_install MySQL-python

mysqld_safe --skip-grant-tables #let anyone have full permissions

mysql -u root

UPDATE mysql.user SET Password = PASSWORD ('nettuts') WHERE User = 'root';
```

#give the user 'root' a password

FLUSH PRIVILEGES;

MySQL -u root -p #log in with our password 'nettuts'

CREATE DATABASE firstblog;

quit

python2.6 manage.py runserver

When you reboot, MySQL won't run, so whenever you need to do this in the future, run MySQL to start the server. You can then run python2.6 manage.py runserver in a new tab to start the development server.

This command will not run the server yet. It will just return an error. That's because we have to adjust our settings. Let's take a look at settings.py.

You must first change the database configuration. These start in line twelve.

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.mysql', # Add 'postgresql_psycopg2', 'postgresql', 'mysql', 'sqlite3' or 'oracle'.

'NAME': 'firstblog', # Or path to database file if using sqlite3.

'USER': 'root', # Not used with sqlite3.

'PASSWORD': 'nettuts', # Not used with sqlite3.

'HOST': '', # Set to empty string for localhost. Not used with sqlite3.

'PORT': '', # Set to empty string for default. Not used with sqlite3.

}

}

If you try to run the server again, it should work, provided you have installed MySQL correctly. If you visit 127.0.01: 8000In your web browser, you should see the default Django page. Now let's turn our Django site into a blog. First, we need to use our Models to create tables in the database by running the following command:

python2.6 manage.py syncdb

Every time you change your models, you must run this command to update the database. Note that this cannot alter existing fields; you can only add new ones. So if you want to remove fields, you will have to do it manually with something like PhpMyAdmin. Since this is our first time running the command, Django will configure all the built-in default tables for things like the admin system. Just write "yes" and then fill in your details. Now let's configure the file urls.py. Write the line:

URL (r '^ $', 'FirstBlog.blog.views.home', name = 'home').

Now, we create the views file to respond to these requests.

from django.shortcuts import render_to_response

from Blog.models import posts

def home (request):

return render_to_response ('index.html')

## Templates

This index.html file doesn't exist yet, so let's do it. Create a folder called templates in your application Blog and save a file calledindex.html, which can simply contain "Hello World" for now. Next, we need to edit the config file, so Django knows where this template is located.

Line 105 is where the section for declaring template folders begins; so adjust it like this:

TEMPLATE_DIRS = (

"blog / templates",

# Put strings here, like "/ home / html / django_templates" or "C: / www / django / templates".

# Always use forward slashes, even on Windows.

# Don't forget to use absolute paths, not relative paths.

)

If you run the server again and refresh the page in your browser, you should see a "Hello world" message. Now we can start designing our blog. We will add some boilerplate HTML for the home page.

<! DOCTYPE html>

<html lang = "en">

<head>

<meta charset = "utf-8" />

<link rel = "stylesheet" href = "css / style.css">

<link href = "images / favicon.ico" rel = "shortcut icon">

<title> First Blog </title>

</head>

<body>

<h1> First Blog </h1>

<h2> Title </h2>

<h3> Posted on date by author </h3>

 Body Text 

</body>

</html>

If you save and refresh the page, you should see that the page has been updated with this new content. The next step is to add dynamic content from the database. To achieve this, Django has a template language that allows you to insert variables with braces. Change the middle section of your page to look like this:

<h1> First Blog </h1>

<h2> {{title}} </h2>

<h3> Posted on {{date}} by {{author}} </h3>

 {{body}} 

We can then pass values to these variable placeholders from the file views.py, creating a dictionary of values.

from django.shortcuts import render_to_response

from Blog.models import posts

def home (request):

content = {

'title': 'My First Post',

'author': 'Giles',

'date': '18th September 2011',

'body': 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam cursus tempus dui, ut vulputate nisl eleifend eget. Aenean just felis, dapibus quis vulputate at, porta et pain. Praesent enim libero, malesuada nec vestibulum vitae, fermentum nec ligula. Etiam eget convallis turpis. Donec non sem fair.

',

}

return render_to_response ('index.html', content)

Save and update, and you should see that you are now transferring content to a template from your views file. The last step is to get data from our database and pass it in instead. Fortunately, we can do all of this without SQL queries, using Django's models. We need to add our application ofBlog to our project FirstBlogchanging other settings. Go toINSTALLED_APPS on line 112 and add to the list the 'FirstBlog.blog', to then change views.py for you to add data from the database.

```
from django.shortcuts import render_to_response

from Blog.models import posts

def home (request):

entries = posts.objects.all () [: 10]

return render_to_response ('index.html', {'posts': entries})
```

Then update the template to access this information.

```
<div class = "container">

<h1> First Blog </h1>

<hr />

{% for post in posts%}

<div class = "post">

<h2> {{post.title}} </h2>

<h3> Posted on {{post.timestamp}} by {{post.author}} </h3>

<p> {{post.bodytext}} </p>

</div>
<hr />
```

{% endfor%}

Here, we can access all the data in our table in the file views.py, then select only the first ten entries. We pass this data to the template, we go through the inputs, and display the data with the HTML of our site. This will not work yet, because there is nothing in the database. Stop the server and run:

python2.6 manage.py syncdb

This will add the new table for our posts to the database. Then open a new tab and type:

MySQL -u root –p

Type your password, press Enter, and run:

INSERT INTO blog_posts (author, title, bodytext) values ('Bob', 'Hello World', 'Lorem Ipsum');

Go back to the previous tab and run the server again. Refresh the page, and you should see a blog post with the fictional content you just added. If you run the MySQL command a few more times, you should see more posts appear on the page when you refresh.

## Django Administration System

The last thing we need to do with this is to check out the Django admin system. This is a powerful feature in Django that allows you to manage your site without writing any more code, just like you would if you were building a site from scratch. To enable it, we need to change some settings. First, uncomment lines 4, 5, 13, and 16 in urls.py, so you can access the administration page.
Then go to section INSTALLED_APPS from settings.py and delete the comment
'django.contrib.admin' and 'django.contrib.admindocs', To allow the administrator to control his table of publications, create a new file calledadmin.py in the folder of Blog and add the following lines:

from Django.contrib import admin

from Blog.models import posts

admin.site.register (posts)

Run python2.6 manage.py syncdb again to add the tables for the admin section and restart the server. If you visit 127.0.0.1:8000/admin now in your browser, you should see a login page.

Use the details you chose earlier when you ran the command syncdbto login. You should see a section called Blog, with a caption for the table of publications. You can use this to create, edit, and delete blog posts with a simple interface. That's all you have to do. You just created a fully functional yet simple blog. To finish this lesson, we are going to see how to install Django on a web server.

## Installation on a Web Server

Most web servers run scripts in various languages using CGI. Django can run on FastCGI and also theoretically CGI, but this is not officially supported and would be too slow for a real production website. You will need to check if they are installed. They are usually found under a heading, such as "CGI and scripting language support."

If you have VPS hosting or are lucky enough to have a dedicated server, your life is so much easier. These usually come with Python pre-installed, and from there, you just need to follow the same steps we went through to get a local copy of Django running. If you don't have Python, you can install it with a package manager. Your system can even come with Django.

ssh root@example.com

wget http://www.djangoproject.com/download/1.3.1/tarball/

tar xzvf Django-1.3.1.tar.gz

cd Django-1.3.1

python setup.py install

Once you have Django installed on your server, upload the site you just created using any file transfer client. You can put the files anywhere, but keep them out of the folderpublic, or anyone will be

able to view the source code of your site. I use /home for all my projects.

Next, create a MySQL database called 'firstblog' on your server and run syncdb again. You will need to re-create your account for the admin control panel, but this is a one-time thing. If you try to run this, you may get an error, and that's because the settings for the server are different from your local computer. You may need to change the database password withinsettings.py, but depending on your server configuration, you may encounter other problems as well. Google is your friend!

To run the server this time, the command is slightly different. You must specify an IP address and a port so that you can access the site over the Internet.

python manage.py runserver 0.0.0.0:8000

If you visit your site in a web browser, on port 8000, you should see your site!

# Final Words

In the Computer Industry, a phenomenon known as free software has been gaining influence. This is a movement that proclaims access to the source code of a program, which admits to being free of use, execution, distribution, and modification. In other words, the new software created under this concept could be used for any purpose, run in any environment, be distributed at the user's own discretion, and be modified if necessary. Programming languages are the basic tool for building programs, as are the machete and hoe for a peasant, the pick and shovel for a builder. Python has been gaining followers in communities such as free, scientific, and educational software due to its simplicity and ability to focus on current problems.

Although this book is not intended for professional programmers, programming at a professional level can be rewarding work, both financially and personally. Creating useful, elegant, and smart programs for others to use is a very creative activity. Your computer or Personal Digital Assistant usually contains many different programs belonging to different groups of programmers, each of them competing for your attention and interest. They all do their best to adapt to your needs and provide you with a satisfying user experience. Sometimes when you choose a certain software, its programmers are directly rewarded for your choice.

Likewise, writing programs is a creative and rewarding activity. You can write programs for many reasons, ranging from being active in solving a complex data analysis problem to doing it for the fun of it by helping others solve a puzzle. This book assumes that everyone needs to know how to program and that once you learn to program, you will find out what you want to do with those newly acquired skills. Python is a language that everyone should know. It is straightforward and clear, and the simple syntax, the dynamic typing, the memory manager, the large number of libraries available, and the power of the language, among others, make

developing an application in Python easy, fast, and, what is more important, fun.

Python's syntax is so simple and close to natural language that Python programs look like pseudocode. For this reason, it is also one of the best languages to start programming. Python is, however, not suitable for low-level programming or performance-critical applications. Some success stories in the use of Python are Google, Yahoo, NASA, Light & Magic Industries, and all Linux distributions, in which Python represents an increasing percentage of the available programs. Always learning a new programming language has new challenges from learning about the philosophy of the language and even its own lexicons, which make the use and expression of its programs with this new language characteristic. But this requires practice and time to become fluent in speaking and writing Python programs. For now, in this book, we learned:

- What is a programming language, and what its fundamentals are.
- The fundamentals of programming with the Python language.
- Working with data types: how to make collections, lists, input and output, and use of variables.
- What classes are and everything that Object-Oriented Programming covers (inheritance and polymorphism).
- The management of files and databases (SQLite).
- The graphical interface for developing visual programs.
- How to document your code and make it an official document.
- How to develop a web page.

There are many useful reports in which we can see that Python is becoming one of the programming languages with the highest growth rate. Python is, statistically, one of the programming languages that has grown the most in recent years. More and more people are interested in learning to program with Python. In fact, there are those who consider it the programming language of the future. A situation that leads to a great demand for specialized professionals. However, this is just one of the benefits of becoming a Python programmer. Becoming a Python programmer is not only

limited to getting more job opportunities but believe us, you are going to have them. It is also about working and specializing in an exciting programming language that you will like more the more you learn about it. Whether you are new to programming or are already a veteran looking for new challenges, Python is for you.

Programming is considered by many a form of art, and like all art, it needs a language that allows you to express ideas in the way you want. Python is the canvas that allows you to reflect, in a simple and elegant way, ideas in an algorithmic way. Its applications, both in the teaching and scientific communities, will allow it to increase its popularity and adoption internationally. Hopefully, this information, especially to the community of geeks around the world, where there are many people, computer scientists or not, interested in taking their first steps in the world of programming in a free software environment, since Python constitutes, without a doubt, one of the best variants.

The learning has not ended here. Indeed, it has only just begun. This book is a tool with every intention of taking you by the hand on your journey as a programmer from day zero. We hope to be a clear, solid, and lasting base of knowledge that will be kept in your collection whenever it is necessary in any challenge encountered in the world of programming. Remember that you can always consult this information again and that a large community of people with the same interests as you is waiting to meet you to help you learn and collaborate.